NUANCE

NEWSLETTER FOR USERS OF ALGOL ON NOVA COMPUTERS AND ECLIPSES

NUANCE 4 1 MAY 1978

From the "Notes" of Dr. Richards:

BIT OPERATIONS.

R. G. Richards

Bit operations can be used to manipulate the individual words. Used in conjunction with their operands, the bit operators lead to the construction of bit expressions. However, these resemble arithmetic expressions and need not formally be distinguished from them.

The general form of a bit expression is:

A. for monadic operators:
   < operator > < operand >
B. for dyadic operators:
   < operand1 > < operator > < operand2 >

In these expressions, an operand may be another bit expression, but in all circumstances it should have the shape of a single (default) precision integer. Thus an operand may be:

a. A default precision integer variable or literal
b. A default precision integer array element
c. A pointer pointing to a single precision based integer
d. A bit expression
e. An arithmetic expression which evaluates to a single precision integer.

Attempts to use operands that are multiprecision give rise to error messages at compile time.

Ther is one monadic bit operator:

NOT < OPERAND >
Result: each 0 (1) bit is changed to 1 (0).

There are three dyadic bit operators:

< OP1 > AND < OP2 >
The result in any bit position is 1 if and only if the bits in the corresponding positions of OP1 and OP2 are both 1 (i.e. if either is 0, the result is 0).

< OP1 > OR < OP2 >
The result is 1 in any bit position if either the corresponding bit in OP1 or in OP2 is 1 (i.e. if both are 0, the result is 0).

< OP1 > XOR < OP2 >
The result is 0 for each bit position if OP1 and OP2 are both 0 or both 1 (i.e. OP1 and OP2 must be different to produce a result of 1).

Note that the bit operations EQV and IMP have not been implemented, but that no error message is produced at compile time if an attempt is made to use them.

[Editor's note: EQV and IMP are implemented for Booleans only, e.g. they will work when "false" corresponds to 0 and "true" to 1, e.g. one bit (position 15) only. In complex bit expressions, an imbedded boolean assignment should be used when these two operators are used. Note also that EQV is identical to NOT XOR, which is implemented for arbitrary bit patterns.]

Further bit operations such as EQV or IMP, can be defined using the OPERATOR procedure.

All possible dyadic bit operations may be mimicked by suitable combinations of the operators provided (in fact, by NOT and any other dyadic operator).

Besides these bit operators, two bit procedures are provided, they are called by the forms:
SHIFT (I,J) and ROTATE (I,J)

These are, formally, single precision integer procedures. The argument I can take any of the forms specified as an operand for a bit operator. The argument J must be an integer variable or literal, negative or positive.

The effect of SHIFT is to move the bit pattern of I by J places:
1. to the left if J is negative; bits passing out to the left beyond position 0 are lost; 0's are introduced at the right as positions are vacated.
2. to the right if J is positive; bits passing out beyond position 15 are lost. 0's are introduced at the left in the positions vacated.

The effect of ROTATE is to move the bit pattern of I by J places:
1. to the left if J is negative; bits passing out to the left beyond position 0 are reintroduced at the right as position 15 is vacated.
2. to the right if J is positive; bits passing out beyond position 15 are reintroduced at the left as position 0 is vacated.

The result of SHIFT(I,16) and SHIFT(I,−16) is 0, that of ROTATE(I,16) and ROTATE(I,−16) is I.

The following notes may be useful:

Neither bit operations nor the bit procedures may be used with multiprecision integers. If it is required to perform these operations on such integers, the user may write his own procedure for doing this; the individual words of the multiprecision integer may be addressed by using pointers.

Since the argument of a bit operator or procedure may be a pointer pointing to a based integer, these operations may be carried out on words assigned to variables of any type. This may be done by obtaining a pointer to the appropriate word of storage using the ADDRESS function and then pointing this at a based integer in the operand or argument. Note that the based variable must be a based integer rather than a based variable of the type of the original variable.

The result of the SHIFT operation is not, in general, identical to multiplying or dividing a binary number by 2. No overflow checking is done and the bit pattern is not even interpreted as a binary number.

/**COMMENT:

ALGOL "OVERSEAS"

For some reason, "overseas" at the East coast of the U. S. almost invariably refers to England [Probably to increase the confusion and misuse of "England," "Great Britain," "U. K.," etc.]. In any case, there is a flourishing, although small, Algol SIG group there, headed vigorously by Dr. E. G. Richards of King's College in London [address, see the list of contributors]. They are holding meetings and share programs and information.

For the use of the local Algol SIG, Dr. Richards wrote "Notes on DGC Algol," the best and most systematic description of the language published to date. It should be required reading for anyone who starts out in this language, and has

continued on page 2

EDITORIAL

The ALGOL - SIG

After a somewhat hesitant start, the Algol SIG (and Nuance, which represents the views of the Algol users) is expanding in membership and activities. There have been Algol meetings during the last three DGC General Users Group meetings, and one is planned again for this fall in Boston. It is clear that Nuance fills a need in user communication. More users are beginning to write about their applications and solutions - rather than only about their initial problems; at the readers request more program examples are given in Nuance. Meanwhile, the Algol UG Library is expanding and being updated, and there are persistent requests for larger articles and papers on specific program areas.

Progress is such that in this editor's view, the bottleneck in further expansion is the editor himself [as proof, this issue originally was planned for Nov./Dec. 1977!] If it is obvious that more users help is needed to increase the usefulness of Nuance and the SIG, it also is an opportune time for feedback from the readers on how to accomplish this. A few questions for which answers are needed follow below, with some tentative and not necessarily optimum answers, to start discussion.

First, is a more formal organization for the Algol SIG needed? At present, Nuance is distributed by DGC to all members of the general Users Group (a superset of the Algol SIG). Active Algolists are writing in and for Nuance, and have no other burdens, duties, or privileges. A case can be made that the extra work associated with formal organization is superfluous, and thus not efficient.

Second, assuming that Nuance still is useful, is it preferable to publish it in shorter intervals? The total volume is set by the amount of text written by contributors: should this be published in one large issue per year, in twelve smaller monthly installments, and on a regular or "random" basis? Experience shows that one large issue is less work, and that the random interval removes one more deadline; but are the readers better served?

Let us assume that 2 to 4 issues per year are optimum. The additional work involved can, in principle, be divided in a number of ways. From the contents, one logical solution would be to have separate editors for the Algol UG Library, for the DOC-BUG section, for letters, and for a DG/L section [this will probably expand considerably when this new language becomes available for more machines.] Additionally, "clerical" work such as editing text, and typesetting is involved in production. (The photo-ready mats are sent to DGC for copying and distribution).

If you have qualifications for any or all the these jobs - or no experience, but just want to learn to help - write as soon as possible: perhaps by the time of the Boston meeting a decision can be made on how to proceed. Perhaps one person with more available time could take over the complete job, or the work could be split; in any case, let your opinion be known even if you cannot volunteer for any of the specific jobs.

A. van Roggen

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, DE 19898.
Tel: 302 - 772.2581

---

continued from page 1

a wealth of useful items for those who think they are familiar with DGC Algol, but then stumble over something unexpected. To quote from the Notes:

"The primary purpose of these notes is to reconcile the manual with the implementation. The notes are written in the form of several short chapters. The reader will find that the same point is noted in several places. This is deliberate, since the author has commonly observed that conciseness is not always compatible with either clarity or usefulness."

With that philosophy, the reader cannot go wrong, and indeed, the Notes are much clearer than the manual. Hopefully, some of this will rub off by the time an official manual update is issued. The Notes are far too long to reproduce here, but the section on Bit Manipulations will serve to give an idea of the style. [If there is interest, other sections can be reproduced in a future Nuance.]

; END **/

CONTRIBUTORS to NUANCE 4

B. Barnett
Faultfinders, Inc.
15 Avis Drive
Latham, NY 12110

S. D. Blessley
Dana Computing Center
Univ. of Hartford
200 Bloomfield Ave.
West Hartford, CT 06117

J. Celko
Box 11023
Atlanta, GA 30310

Dr. H. W. Eber,
Psychological Resources
1422 W. Peachtree Str. NW
Atlanta, GA 30309

R. Fessenden,Hague International
3 Adams Str.
S. Portland, MN 04106

N. Finn
ROLM Corp.
4900 Old Ironsides Dr.
Santa Clara, CA 95050

B. Friedlander,
A. D. Little Systems
Burlington, MA

S. M. Heidel
265 Freeport Road
Pittsburgh, PA 15238

J. Huffman,
Data General Corp.
Westboro, MA 01581

G. D. Jelatis
U. of Minn. Hospitals
420 Delaware Str. SE
Minneapolis, MN 55455

B. Johnson,
Data General Middle East
260 Syngrou Avenue
Athens, Greece

D. R. Kaye
Du Art Film Labs Inc.
245 W. 55 Str.
New York, NY 10019

N. M. Kittredge
Pacesetter Corp.
Marsh Road
Litchfield, CT 06759

Dr. P. Maas
Dept. Appl. Physics,
U. of Strathclyde
Glasgow, Scotland U. K.

J. Maloney,
Data General Corp.
Westboro, MA 01581

J. Miranda
Centro de Calculo de la
Universidad Politecnica
Barcelona 14, Spain

Dr. E. G. Richards
Dept. of Biophysics
King's College
26-29 Drury Lane
London WC2 England

M. Ruggera
Dept of Chemistry
U. of California
Irvine, CA 92715

W. D. Selles,
New England Med. Center
171 Harrison Ave.
Boston, MA 02111

F. V. S. Shafer
RCA Astro Physics
PO Box 800
Princeton, NJ 08540

Dr. T. K. Sharpless
Memorial Sloan-Kettering
Cancer Center
1275 York Ave.
New York, NY 10012

P. G. Smith
Dept. of Psychology
Stanford University
Stanford, CA 94305

A. J. Thomas
Institut d'Anatomie
Universite de Lausanne
Lausanne, Switzerland

C. Waters
RK Steedman & Assoc.
384 Rokeby Road
Subiaco 6008
Western Australia

I. S. Wolfe
PO Box 1092
Longview, Wash. 98632

The New DGC Algol: DG/L

J. Maloney and J. Huffman

Data General has recently announced the availability of DG/L, a programming language based on DGC's Algol. This new language was developed internally by DGC, and is designed for the implementation of a broad range of applications across the entire DGC hardware/software product line. It is suitable for heavy computation jobs as well as more "data processing" functions. DG/L comes equipped with a multitasking runtime environment which, in conjunction with its string handling and string arithmetic capabilities, make it an ideal tool for the development of multi-terminal on-line applications for the Nova and microNova product line, as well as for Eclipses, on which the compiler itself operates.

The operating system interface, virtual data structures, address manipulation, and recursive procedure definition features make DG/L an extremely useful tool for the development of system level software such as compilers, assemblers, sort/merges, and other utilities. Designed as a highly compatible superset of DGC Algol (the differences are documented in the DG/L manual), DG/L is ideal for the use of structured programming techniques and variable scoping, as well as extensive string management functions that are well suited for many commercial applications.

Additional features include:

1. Algol-like syntax, block structured, procedure oriented.
2. Powerful memory management techniques, including dynamic allocate/free and cache memory handlers for virtual structures.
3. Standard single and double precision integer arithmetic, optional single and double precision real arithmetic.
4. String manipulation and string arithmetic.
5. Direct address manipulation and based variables.
6. Globally optimizing compiler; generated code is reentrant and recursive.
7. Multitasking runtime environment.
8. Compatibility across the entire line of DGC hardware and software. One compiler can generate code for execution under AOS, Eclipse RDOS, Nova RDOS, DOS, and RTOS, and microNova, from the same source file.

Perhaps the most attractive feature of the language is its compatibility: the source language syntax includes a comprehensive operating system interface. The programmer codes in one form, regardless of the target system, and the compiler and runtime library provide the proper object system program. A programmer must compile on an Eclipse RDOS system, and can target his code to any RDOS, RTOS, or DOS system. Using an AOS program development system, a programmer may write, compile, and debug his code under AOS and then create the RDOS, DOS, or RTOS save file and dump it on tape or diskette for transport to the target machine. The compiler uses switch options to control code generation for AOS, Eclipse RDOS, etc. The proper system libraries (ASYS, BSYS, NSYS, etc.) must be loaded on the AOS disk, and with the AOS utility "RDOSBIND," an RDOS save file can be made.

END;

ADVANCED LOOP—MANSHIP

J. Celko

I will presume that the reader already knows that a loop is a programming device used to execute repetitively a set of program statements, and that loops are used to simplify and shorten programs. Some Algol formalisms and tricks are described below.

The basic form of the loop statement is:

FOR <cv> := <elem.list> DO <st>

where <cv> = controlled variable, <st> = a statement or compound statement or block, and <elem.lst> = a list of values and expressions, used for the controlled variable. FOR loops in Algol are very general structures compared to loop constructs in other languages; this flexibility lets the Algolist get a good bit more out of his language than the BASIC or FORTRAN programmer.

THE ELEMENT LIST.

One form of the FOR loop consists of a list of elements of the same data type as <cv>, separated by commas. The trick here is that they do not have to be integers as in Fortran; try using Real variables sometimes. But an even better use is that in a string list; this is an extension in DGC Algol, and not a part of the Algol-60 standard. String lists are very handy for text work, since the strings can be of different lengths.

```
FOR R:= 0, 1.5, 4.237 DO
  BEGIN CIRC:=2*PI*R;WRITE(0,CIRC) END;
FOR ST:= "A," "ALFA," "THE END" DO STRPROC;
```

where STRPROC is a procedure operating on the strings.

The step-until element.

In the revised Algol-60 report a step-until element is defined as

```
V:=A;
L1: IF (V-C)*SIGN(B) > 0 THEN GOTO EE;
  S1;
  V:=V+B;
  GOTO L1;
EE: /*element exhausted*/
```

as all old-timers will remember from their school days. The principal goodie you got to use here was that the step size could vary with the execution of the loop. As an exercise, try writing a binary search with one for-loop whose step size keeps getting half sized, and whose sign changes with each execution. [And has a properly defined exit! See also Nuance 3, p 9, the "Time reversal problem," for an unusual loop. Ed.]

This sort of changing variable is rough on a compiler and can cost you some overhead because the machine cannot simply use register increments for the control variable. Also, in the Revised Report, the control variable kept its last value if the loop was exited through a goto, but it was undefined if it was exited by exhausting the for-list elements. DGC Algol will save the last value in all cases, but this is not standard.

The Modified Algol Report from IFIP working group 2.1 has some subtle differences that the complete Algolist should know. First of all, the for-loop step-until element is defined as:

```
FOR V := A STEP B UNTIL C DO S;
```

which acts as

```
BEGIN <data type of B> BB;
V := A;
LB: BB := B;
  IF (V-C)*SIGN(BB) <= 0 THEN
    BEGIN
    S;
    V := V + BB; GOTO LB;
    END;
END;
```

In short, the step size is held constant, and in addition, the control variable will retain the last value assigned to it on either exit condition.

The effect of a goto leading into a loop body (S, in the above example), was undefined in the Revised Report, but in the Modified Report the body of the loop is taken to be a block, so all labels are local to it, and cannot be used from some statement outside of the body. But of course, nobody writes goto's any more, so this does not apply to good Algolist!

Most readers should know that a step-until construct is better than using a list, because of the use of machine features to generate the element list.

WHILE—LOOPS.

It is pretty clear to even a beginner that he can write the step-until element of a for-loop in terms of a while-loop construct directly from the definitions given in the last section of this article. Don't do it in most cases: the step-until is going to run faster.

A while-loop is useful if the termination of the loop is due to other factors than the value of the loop control. For example, in a linear table search, you would stop the loop because the target has been found or because you came to the end of the table. One way to write this would be:

```
FOR I:=1, I+1
  WHILE (NOT EDT AND NOT FOUND) DO S;
```

A little more polished method is McCarthy's logical operators in Algol form. Let us suppose that we have a very large table, and we expect to find almost every value we are looking for, so that the loop will terminate on a found condition most of the time. We could then write:

```
FOR I:=1, I+1 WHILE
  (IF FOUND THEN TRUE ELSE NOT EDT) DO S;
```

The boolean can be adjusted a bit further to get rid of the negations by using boolean algebra:

```
IF FOUND THEN FALSE ELSE NOT EDT
```

While this is messy in one way, it buys some speed in the cases where the compiler does its logical work by bit commands (as in DG's case). In an optimizing compiler this would be done as part of the compiling.

Structured programming people will remember that there are two classic program loops, named the While-loop and the Until-loop. These loops are also named Pre-test and Post-test loops, to avoid confusion with the Algol keywords. We can make these classic loops in Algol with "while." For a pre-

continued on page 4

continued from page 3

test loop, use:

FOR I:=I WHILE (B) DO S;

and as a post-test loop:

FOR I:=I, I WHILE (B) DO S;

The post-test loop is the same thing as the Pascal "Repeat S until (not B)" construct. That negation might throw the Pascal programmer off a bit at first!

END;

ALGOL INPUT/OUTPUT DATA CONVERSION PROBLEMS

P. Maas

The Algol READ routine (Rev.: RDOS 5.00, Algol 2.03) makes very few tests for compatibility between external data and internal variables. Even worse, it often makes unwarranted assumptions about conversions to be made. And since data conversion does not cause an exit to an error label, there is no way for a user to validate his input.

In the examples given below, the following declarations hold:

```
BOOLEAN B;
INTEGER I;
REAL R;
```

and EOFL and ERRL are labels to which transfer is made when an end-of-file, respectively an input error, is found.

First, consider the statement

```
READ(0,B,EOFL,ERRL);
```

Any input string beginning with character "T" is treated as equivalent to the strings "T" and "TRUE. " Any other input string beginning with a letter differing from "T" is treated as equivalent to "FALSE. " Worse, if the input is a number, the variable is set to FALSE and no error indication is given.

Next, consider the statement

```
READ(0,I,EOFL,ERRL);
```

If input is an integer which is in range, all goes well. If input is an integer out of range of the 1-word precision, (such as 44444), the error is trapped as an integer conversion error, but the error return is not taken. If a real number is given (such as 1.5), the input error is untrapped, and the integer is set to the integer part of the input number. If input is an alphanumeric string, the conversion error is not trapped again, and a value built from the numerals in the string is assigned to the variable!

Finally, consider the statement

```
READ(0,R,EOFL,ERRL);
```

If the input datum is real and in range, all goes well. However, if the datum is real and out of range (e.g. 1.0E−78), an arbitrary value is assigned to the variable and no error return taken. If an integer datum is supplied, it is "widened" to the corresponding real value and assigned correctly to the variable. Finally, if an alphanumeric string is input, no error indication is given, and the real variable is set to zero.

For all three versions of the READ statement, if the last datum in the input file is followed immediately by the end-of-file (EOF) and not by any other field terminator, (e.g. OR, LF, FF, TAB, SPACE, etc.), then the last datum is not read. Instead the READ routine immediately jumps to the EOFL label.

What might we ask DGC to do about this? In theory, the user can write his own input routines using BYTEREAD and LINEREAD, and do his own conversions and data validation. This does involve many users in additional programming effort, and in extra I/O overhead in all programs.

If DGC wishes to reserve ERRL for hardware I/O errors, then the READ routine might be rewritten to provide a third error label return for data conversion errors. Return to such a label should be made whenever an unacceptable datum is encountered. In addition, an error message should be provided, giving the specific form of error.

What should be the acceptable input data for each data type? I think the following limits should suffice:

```
BOOLEAN T to TRUE, F or FALSE only
INTEGER any in-range integer only
REAL any in-range real or integer only
STRING any string < max. length
```

(in-range implies the precision of the variable for which the assignment is made.)

Since Algol is structured to encourage both careful and defensive programming, it is well suited to the production of user generated utility programs. Utility programs are robust only if the programmer is able to verify his input (and thus defend his program against "garbage in"). At present, the READ procedure provides the user with little opportunity to defend his program. It would significantly improve life for us all, if a revised READ procedure were produced for the runtime library by DGC.

So let me finish with a plea to DGC software development and maintenance groups... "You have a good product in Algol, together with users interested in improving the product. Please, take my advice, and start fixing it up. If you cannot devote corporate time and money to the problem, why not release the source code version of the compiler and runtime package to the Algol users SIG so we can fix it for you?

[Editor's note: One other problem with READ of strings, is that a conversion takes place from lower to upper case. This makes word-processing applications rather awkward. For more on upper/lower case handling, see D5 in this issue's DOCBUG.]

END;

ALGOL UG LIBRARY UPDATES

In addition to the library programs listed in Nuance 3, new programs are being announced. Nuance will publish descriptions of such programs that are written in Algol, or useful with Algol. There also are updates of older programs being submitted; for example, in the spirit of ternational cooperation, Alex and Alglib are being updated in a combined effort from Scotland and the US (to incorporate more error checking and more flexible operation, and a version of Alex for Fortran programs is on its way).

MAKEPRETTY

A copy of this program was sent to Nuance by the author, N. Finn. The program takes as input, an ASM or MAC source file, and reformats this text into a better readable one with uniform appearance. In the same time, it manages to correct mistakes in the input file (unfortunately, only typing errors, not faulty programming) and checks for format legality in ASM or MAC. For example,

MAKEPRETTY/M X.SR XN.SR 30/C 19/N

tests file X.SR according to MAC rules, reformats it with opcodes starting at column 19, and comments at column 30, and puts the result into file XN.SR. On the sample programs provided, the change was incredible. True, the input file was not a real program (hope nobody makes that messy a job) but a test text. Double commas were removed, the indirect sign moved to a standard location, depending on the code [JMP Q2,3 but LDA 2,Q3], double labels put on consecutive lines, and - as was to be expected, everything was lined up properly. Those who use ASM only as a patch to Algol (and thus usually as a compiler generated source) may not have much use for the program Makepretty, but once you have to write device drivers to make your favorite gadgets Algol callable, or other ASM or MAC programs, Makepretty will come in very handy.

The program compiles and runs properly (Nova MRDOS) with a size of 15.3 kbytes for the .SV file.

XXREF Cross - References

This program was contributed by F. V. S. Shafer, and operates on a standard RLDR command line, e.g.

RLDR FILE1 FILE2 LIBR.LB

If this string is used (with or without the "RLDR") as argument for XXREF, the program causes a three-part output. The first part is a listing in load line sequence of all program modules, their entry points, and external references:

```
.TITL=   DEV66
.ENT=    T70 L41 C70 DOFRE
.EXTN=   ERIN .XMT .REC
(etc.)
```

The second part of XXREF's output is the cross reference listing by module name. This contains all entry points which RLDR would have loaded, cross referenced with all modules that refer to them:

TITL	ENTRY	REFERENCED BY
RTI	RTI	—
AREC	AREC	DASSE
BGICM	TIMGI	FGSEN
RTSPR	USERD	TASKS
RTCHR	SC	CVWOR TASKS
CLOSE	CLOSE	TPU RTSIO RISIN
(etc.)		

The third part of the output contains a list of unsatisfied externals and the modules that use them.

Various switch options are possible, e.g. the inclusion or omission of library listings, the systems library, etc.

Use of this program is highly recommended for any system consisting of large programs with overlays, external devices with their drivers, and large numbers of external procedures. Even when used not as a debugging tool, the program is of great value in getting a proper systems documentation.

[From program description only, no run tests made.]

A REPORT ON THE ALGOL-SIG MEETING

San Francisco, 28 Oct. 1977

The Algol SIG had a well-attended session in San Francisco during the DGC Users Group meeting. The session started with a short report on the current situation in Algol. The increase in Algol activity is reflected in the number of programs submitted to the Users Library (See Nuance3). A request was made to the users to submit more written articles, etc. on Algol, specifically program examples, solutions to problems, questions and needs in Algol, etc. Currently, Nuance is being distributed by DGC, due to efforts by Dale Silva, John Maloney, and others who work in the User Group support. Their support is appreciated by the Algol users.

The next item on the agenda also indicates that Algol is flourishing: Brad Friedlander announced that the next revision of Algol (Rev. 2.10) is in production, and will be released soon. In this Rev, all integer arithmetic bugs have been cured, as well as the global "goto," the bug in referencing global variables and labels, in blocks that are two or more levels up.

An application of the CMM (cache memory management) package was described by A. van Roggen. This package from the DGC Algol extensions, is available in Algol and in DG/L, and is very useful for database manipulation. Examples of database structures were shown with the corresponding binary trees. It was then shown how links could be set up (in addition to those pertaining to the tree structure), so that items from the database can be accessed much faster than from regular files with sequential reads. This difference in speed can be crucial: a small database with 80 sequential items, each having 13 attributes, takes about 45 seconds for each "edit" operation. The binary tree method takes about two seconds. (An updated version, based on tuples, rather than binary trees, is even faster: more than 100 items with 18 attributes is "instantaneous" relative to the console writing speed.) The CMM procedures handle files essentially as an extended virtual memory. If the user sets up a strategy, such as the tuples or tree structure mentioned, whereby only 2 file reads have to be made to access any item in the database, DGC's extensions improve this to "a maximum of 2 file reads," e.g. to 1 or even 0 file reads: files are read in blocks, and active blocks remain longer in fast memory.

The last part of the Algol meeting was taken up by B. Friedlander, who introduced the new DG/L, with emphasis on its technical features, the differences with Algol60 and DGC Algol. Most Nuance readers will be familiar with DGC Algol; DG/L essentially is an enormously extended version which "solves" almost all of the problems that users have encountered in Rev2.03 Algol. DG/L's code runs much faster, due to optimal compiling; a choice can be made between integer and real results in division of two integers; an even further extended looping statement is provided; multitasking is fully supported; string operations are extended; the regular Fortran math package can be used; and procedures exist for executing machine language instructions immediately from the DG/L program. For all these improvements, a few operations from DGC Algol have been dropped. These include the extended (up to 15 word) precision of integers and reals (only single and double precision are supported), and the ability to define operators (but a string concatenation operator is now included.)

Another useful feature in DG/L is its conditional compile: a statement can be either compiled or not, depending on the setting of a switch during compilation. Other methods of commenting are implemented, to facilitate even further the in-program documentation inherent in Algol. New conditional expressions are WHILE B DO S; and DO S WHILE B; with equivalent statements using UNTIL instead of WHILE.

Brad mentioned that he had updated all his Algol programs to DG/L and that the effort was well worth the little effort, because of the increase in speed and the extra features. Regular Algol programs hardly need any change to be compileable in DG/L - you have to watch out only if you have used special ASM procedures or other "tricks" in your Algol programs.

From the audience response, it was apparent that DG/L is the way to go, and those who have Eclipses probably are switching over. The main item that will hold the majority of Algol users back is that DG/L, although it can run on any DGC machine, including the microNova, it can be compiled only on an Eclipse. In the Software Panel discussion, this problem was discussed, as well as the way DG/L will be released to the users. Hopefully, a DG/L version for Novas will be announced soon.

At the end of the presentation, there was a rush to the front desk where many names were added to the list of people requesting a copy of the DG/L manual (which had not yet been printed). In all, an enthusiastic ending to a worthwhile Algol SIG meeting.

A. van Roggen

One simple, and one flexible procedure to shift upper/lower case strings. See DOC-BUG, D5.

```
PROCEDURE UPSHIFT (S);
STRING S;
/*Shifts string to upper case*/
BEGIN INTEGER I,L,A;
L:=LENGTH(S);FOR I:=1 STEP 1 UNTIL L DO
 IF (A:=ASCII(S,I))>96 AND A<123 THEN
  BEGIN A:=A-40R8;SUBSTR(S,I):=SUBSTR(A,2) END;
END UPSHIFT;

PROCEDURE CAPSHIFT(S,F);
VALUE F;STRING (130) S;INTEGER F;
/*Shifts strings to upper or lower or mixed case.
With F<1 gives all lower case, F>1 all upper case.
When F=1 the first letters are capitalized of all words.*/
BEGIN LITERAL AU(65),AL(97),ZU(90), ZL(122);
INTEGER L,I,C;BOOLEAN SPC,LC,UC,FRST,UP,DN;
  PROCEDURE CASE;
  BEGIN
  LC:=C>=AL AND C<=ZL;UC:=C>=AU AND C<=ZU
  END CASE;
L:=LENGTH(S);SPC:=TRUE;FRST:=F=1;UP:=F>1;DN:=F<1;
FOR I:=1 STEP 1 UNTIL L DO
  BEGIN
  C:=ASCII(S,I);CASE;
  IF (LC AND UP) OR (LC AND SPC AND FRST) THEN
    SUBSTR(S,I):=SUBSTR((C:=C-40R8),2)/*up*/
  ELSE
  IF (UC AND DN) OR (UC AND FRST AND NOT SPC) THEN
    SUBSTR(S,I):=SUBSTR((C:=C+40R8),2)/*down*/;
  SPC:=C=32/*space*/
  END;
END CAPSHIFT;

END;
```

LETTERS:

FROM: T. K. Sharpless, Sloan-Kettering.

Should we be working for a more efficient realization of Algol on the NOVA computers? Or does it intrinsically cost three times as much to use Algol as Fortran?

I enclosed results of a simple speed test in which the Basic, Fortran, and Algol current to RDOS Rev 5.00 were compared without and with the aid of integer mult/div hardware on a NOVA 1220 (Programs below). The results were certainly not flattering to DGC's implementation of Algol. Particularly striking is the fact that the absolute time savings in using hardware M/D was twice as great for Algol as for Fortran and Basic, implying that Algol uses twice as much arithmetic to get the answer. While there may be some specific defect in FLOAT or COS, I suspect that the runtime overhead (linkage and subscripting) are equally at fault, since I have consistently seen disappointed in the speed of Algol for compute-bound applications.

Much of the benefit of a rational, standardized language disappears when one is forced to learn to "code around" defects in an implementation. I would rather devote that kind of ingenuity to ASM programming.

```
BEGIN
EXTERNAL PROCEDURE GTIME;
REAL ARRAY A[ 100,5];
INTEGER I,J,K,Y,MO,D,H,M,S,T;
OPEN(0,"$TTO");GTIME(Y,MO,D,H,M,S);
TL=S+60*(M+60*H);
FOR K:=1 STEP 1 UNTIL 10 DO
  FOR I:=1 STEP 1 UNTIL 100 DO
    FOR J:=1 STEP 1 UNTIL 5 DO
      A[I,J]:=COS(FLOAT(I)/FLOAT(J));
GTIME(Y,MO,D,H,M,S);
T:=S+60*(M+60*H)-T;
WRITE(0,T," sec for 5000 cosines<15>");
CLOSE(0);
END;
```

continued on page 6

continued from page 5

```
      DIMENSION A(100,5)
      INTEGER T
      CALL FOPEN (0,"STTO")
      CALL FGTIME (IH,IM,IS)
      T=IS+60*(IM+60*IH)
      DO 10 K=1,10
      DO 10I=1,100
      DO 10 J=1,5
      A(I,J)=COS(FLOAT(I)/FLOAT(J))
10    CONTINUE
      CALL FGTIME(IH,IM,IS)
      T=IS+60*(IM+60*IH)-T
      WRITE(0,100),T
100   FORMAT(IH,I5,"SEC  FOR 5000 COSINES"/)
      END
```

[The Basic program was not provided.]

TIMING TABLE (sec)

LANGUAGE	HW M/D	SW M/D
Algol	127	153
Fortran	22	34
Basic	62	73

My most pressing software problem is how to provide a "time sharing" environment for interactive data collection and analysis, on a system of 32K NOVA line processors, linked by MCA units (DGC's best hardware idea), one of which is a "backend" machine devoted to maintaining a file system (4x2.5 Mbyte disks, magtape, printer) under RDOS. The "frontend" machines require very flexible facilities for filing and retrieval, including dynamic procedure loading and linking, and multi (well, at least two —) user support with considerable multitasking. RTOS is the obvious executive, but would need such extensive fortification that a standalone system might be as easy to build. Do you know of anyone with relevant experience, software, or free time?

Keep the newsletter coming; with better typography it would be a real literary pleasure.

/**COMMENT:
For more details on timing of floating point operations, see the letter from B. Johnson, this issue.
;END **/

FROM: M. B. Ruggera, U of CA, Irvine.

We have a NOVA3 with a Tektronix 4006-1, Versatec 1200A printer plotter, 4234 top-loader disk, and high-speed paper tape reader. We are running under RDOS5.00, Algol2.03. In this environment, I have discovered one peculiarity and one bug which brings down the system when programming (forgive me!) games.

The peculiarity is demonstrated in a program which generates mazes with only one path through. Part of this program goes as follows:

```
OKLEFT:= ...;OKRIGHT:= ...;
OKUP:= ...; OKDOWN:= ...;
/* GOTO XXX; GOTO GOUP; GOTO GODOWN;
GOTO GORIGHT;GOTO GOLEFT;
XXX: */
IF OKLEFT AND OKUP AND OKRIGHT THEN
  BEGIN
  SWITCH XYZ:= GOLEFT, GOUP, GORIGHT;
  GOTO XYZ[RAMD3]
  END;
; ...;
GOUV: ...
GORIGHT: ...
GOLEFT: ...
```

This program, with the /* */ comments as shown, gives a set of compiler errors:
"Use of incorrectly declared variable" for the labels GOLEFT, GOUP, GORIGHT. However, if the labels are used earlier, e.g. by removing the comment markers, (and then, of course, bypassed by using the dummy label XXX), the program compiles and runs correctly.

The bug which blows the system occurs in a program DUKEDOM, which uses a large set of string literals. In compiling, the system bombs with the utterance "Statement does not end properly" at a very proper statement. I attempted compiling this program at our local (Santa Ana) DGC office. There, the program did not bring down the system, but reported "E21," a compiler error. We notified DGC about this more than a year ago, but have not had a response from them. My feeling is that this error is related to the number of literals: by removing the literal strings, the error can be moved about.

I was very happy to receive Nuance. It is reassuring to know that I am not the only person having problems with this compiler. I have encountered most of the bugs described. Could you give information on how to submit programs for the Users Group?

/** COMMENT:
See the DOCBUG section for a possible bypass of the excess literal problem, and for another contribution by Ruggera. The unrecognized GOANYWHERE labels of the example probably are fixed in Algol 2.10. The current version cannot find some variables in global blocks. The final question is easy to answer: the most difficult thing in contributing programs is to write them, and to provide enough documentation that people can run them - and learn by studying the code. After that, just send the program to the Users Group at DGC in Westboro Mass. At the same time, a description of the program should be sent for inclusion in Nuance (address, see p.2) if the program is in Algol or has Algol related applications.
; END **/

FROM: B. Barnett, Faultfinders Inc.

After reading the November 1977 issue of Focus, I would like to express a great desire to join your Algol SIG. I represent a group of 10 programmers. We primarily use RDOS Fortran, Algol, and ASM. In two months, we will get an AOS system with DG/L. With the new system, we will use Algol and DG/L for systems development.

I hope we can help each other through the Algol SIG.
; END **/

FROM: S. D. Blessley, U. of Hartford.

We are interested in membership of the DGC Algol SIG, either as individuals or as an organization. Most here have found Algol to be a frustrating and uncooperative language to work with - only myself and a few brave souls here at DANA have gotten DGC Algol to work for us, rather than we for it.

I received the third issue of Nuance; it was nothing short of fantastic in my opinion. Myself and others would like to acquire previous and future issues if possible.

Our experiences with DGC have been short of desirable in some cases, particularly in regard to communication with the Users Group in nearby Westboro, Mass. I presume the SIGs are independent of their organization.
; END **/

FROM: N. M. Kittredge, Pacesetter Corp.

I enjoyed reading Nuance3. Please send me back issues of the newsletter and place me on the mailing list. Sometime in the near future, I hope to contribute an article.
; END **/

FROM: J. Miranda, Un. Politecnica Barcelona.

Our computing center for the Technical University at Barcelona, Spain, is interested in receiving information on using Algol on our S-200 and NOVA 2/10 computers. Please add us to your mailing list.
; END **/

FROM: A. J. Thomas, U. de Lausanne

Please put me on the mailing list of Nuance. I am using Algol on a NOVA for image processing of electron micrographs, and am delighted that from now on I will not have to discover the pleasures and (considerable) perils of DGC Algol on my own!

/** COMMENT:
The above letters are typical of the requests that come in for Nuance. Since issue 3, the DGC Users Group is distributing Nuance with their Focus, and thus all that is needed to get future Nuance issues is to make certain that you are on the Users Group list. Of course, the generation of future issues depends heavily on contributions (in writing!) from users; this is also true for Focus which needs more active users support in areas of technical articles and applications.
; END **/

continued on page 7

continued from page 6

FROM: D. R. Kaye, Du Art Film Labs

We are very heavy users of DGC's Algol and also of our own language Procal (Process control Algol) which uses a much modified version of the DGC compiler. We are finally ready to report our results with the compiler and library modifications. Additionally, we are evaluating DGC's DG/L compiler and would be willing to write a short "review" of the language and its compiler.

/** COMMENT:
Nuance readers will not only be interested in a review of DG/L, but also in Procal, its availability, and the design criteria that have gone into this effort. There are a number of fairly similar languages (e.g Pascal, C) each with its own "philosophy" and utility. Familiarity with several of these makes for better programmers.
; END **/

FROM: B. Johnson, DGC Athens, Greece.

/*** See "Matrix Inversion," Nuance3, p.15 ***/

Where does the array index checking time go? You came close to an accurate answer in the following sentence. If you had said "Is it negligible with the overhead in software floating point calculations?", the answer would be yes. A compute bound program running under RDOS incurs less than 5% overhead penalty if no system (.SYSTM) calls are being executed and the clock is sysgen'ed for 10 Hz. I am assuming an 800 series processor. The effect you were seeing was the fact that subscript checking takes on the order of 10's of microseconds, and software floating point emulation (especially with the variable length mantissa code our Algol uses) takes around 2 to 10 milliseconds. This is thus a factor of 200 to 1000. Unless your use of subscripted variables exceeded your use of floating point operations to a considerable degree, you should not expect to notice the penalty. In default integer programs, however, the effect can be monstrous.

/** COMMENT:
Data from the (Trojan??) horse's mouth! This is the type of useful information which is not in the manual and normally very difficult to get (or time consuming to find out the hard way). It implies that the extra effort needed to set up pointers, compared to direct array usage with subscripts, is not needed in most cases where REALs are concerned, but should be considered in some INTEGER array operations.
; END **/

FROM: S. M. Heidel, Pittsburgh

Just a note to say that I have enjoyed reading the past few issues of Nuance, which are very interesting.

You seem to be interested in getting more programs into the DGC Users Group Library; however, it takes them forever to announce a program in a newsletter. It took them literally several years to announce one of my programs, and they still have not released another one. I have one more program that is almost ready, however, it hardly seems worth the effort. I suspect that things will get worse now that Tom Streck is no longer running the users group. I may be wrong, but I also get the impression that DGC is not really interested in the scientific user these days. The sales people I meet from DGC do not seem to know much about either DGC's or DEC's offerings.

/** COMMENT:
I hope that the delays are a trauma of the past: there has been a vast improvement the last year, with a new catalog of programs, and a more systematic method of listing and ordering. I do not know how frequent the catalog will be updated, but one sure way to beat this is to write a description of the program with examples, etc. for Nuance, or for Focus if the program is not Algol related.
; END **/

FROM: P. G. Smith, Stanford U.

I enjoyed the Database talk at the Users Group meeting in San Francisco, and can use this type of programs.

With stimulation from Nuance, I have finished my first Algol program, after 15 years of Fortran. The newsletter is of great help.
; END **/

FROM: R. Fessenden, Hague Int'l.

More on Entier

In order to maintain correct precision to the nearest penny on REAL (4) numbers in our commercial applications, we needed a working function like ENTIER. Unfortunately, the fixes suggested so far in Nuance are only good for default precision. Having concluded that Algol provided no help, we turned to ASM in desperation.

The first thing we did was to compare statements identical in all respects, except for the Entier function. Two such statements are shown in Fig.1a and 1 b.

FIGURE 1a

```
    ; KEEP := B + C;
11 JSR          QBLKSTART
12 FENTL
13 FPRC         4
14 FLDA         0,S+4,3          ;B
15 FLDA         1,S+10,3         ;C
15 FADD         0,1
17 FEXT
20 XENTL
21 FPRC         2
22 FSTA         1,S+16,3         ;KEEP
```

FIGURE 1b

```
    ; KEEP := ENTIER(B + C);
11 JSR          QQBLKSTART
12 FENTL
13 FPRC         4
14 FLDA         0,S+4,3          ;B
15 FLDA         1,S+10,3         ;C
16 FADD         0,1
17 FEXT
20 XENTL
21 FMOV         1,1
22 FPRC         2
23 FSTA         1,S+16,3         ;KEEP
    ; DECPART := KEEP;
24 FEXT
25 FENTL
26 FPRC         4
27 FSTA         1,S+20,3         ;DECPART
```

Note that in Fig.1a, B and C are real (4), while KEEP is integer (2). Furthermore, in both Figs.1 a and b, the above statement was followed by DECPART := KEEP;. The statements shown in Fig.1 are exactly as compiled in both cases. DECPART is real (4). In both examples, the program works incorrectly! First, in 1b, the FMDV 1,1 did nothing, and "explains" why entier does not work. However, even 1a failed to truncate properly. The result of 1a is the same as if it were coded DECPART := B+ C; KEEP := DECPART. The reason is clear: (a) the contents of register 1 contain the result from adding B and C, (b) Register 1 is sent to KEEP, where of necessity it becomes integer (2), and (c) register 1 is sent to DECPART, where it remains unchanged! The code inserted after statement 24 in 1b will properly truncate the real (4) number to an integer (2) form. If you really want an Entier function that works, you can follow van Roggen (Nuance 2, p.17), with only minor variations:

```
INTEGER (2) PROCEDURE ENTIER(R);
REAL (4) R;
BEGIN ENTIER:=R;
IF R<0.0P4 AND ENTIER<>R THEN
  ENTIER:=ENTIER - 1;
END ENTIER;
```

Now - can anyone tell me how to make the above procedure to work for any precision, as the one in the book is supposed to?

/** COMMENT:
Has anyone succeeded in manipulating the argument precision, or alternatively, retrieving the precision of a procedure argument inside the body of the procedure? Only literals are allowed as precision specifiers, and it is not immediately obvious how to retrieve the precision.
; END **/

FROM: W. D. Selles, N. E. Medical Center

I have been looking for a good document-formatting program to run on a NOVA, and remember that Nuance was prepared with such a program. Is this available to other users?

/** COMMENT:
The early nuances were prepared on a timeshare system with their resident

continued on page 8

continued from page 7

formatter. However, since last year, the program SCRIBE has been available from the DGC Users Group; this is the program that is now used to format the text. The only thing missing for Nuance is a good and readable printer.
; END **/

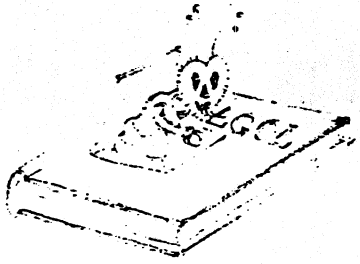
FROM: G. D. Jelatis, U. of Minn.

I received a copy of Nuance 3 with Focus, and it renewed my interest in the Newsletter. Would you please add me to your mailing list and send me a copy of Nuance 2, (somehow, I got a copy of Nuance 1)?

I am starting to use DGC Algol and hope to implement some of the utility software routines described in Software Tools (a la Unix) in Algol, which I believe is a better choice for such things than Ratfor.
; END **/

A procedure to accept upper and loser case from a console. See this issue, DOC-BUG D5, and Nuance 3 p.4, under ALGPROC. Use CONSOLE to define CNSO, CNSI.

```
STRING (130) PROCEDURE ASKS(L,M);
VALUE M;INTEGER L;STRING M;
/*Formatted string input from console, types string M,
  returns bytelength L, and reads upper/lower case.
  The DEL/RUBOUT key echoes and deletes previously typed characters, as
in EDIT.
BEGIN LITERAL CR("<15>"),RUB("<177>");
EXTERNAL INTEGER CNSO,CNSI;
POINTER P;STRING (1) C;STRING (130) SL;
WRITE(CNSO,M);L:=0;SETCURRENT(C,1);P:=ADDRESS(C);
NXT: BYTEREAD(CNSI,P,1);IF C=RUB THEN
  BEGIN
  WRITE(CNSO,SUBSTR(SL,L));SUBSTR(SL,L):="<0>";
  IF (L:=L-1)<0 THEN L:=0;GOTO NXT;
  END ELSE
  BEGIN
  WRITE(CNSO,C);IF C<>CR THEN
      BEGIN SUBSTR(SL,(L:=L+1)):=C;GOTO NXT END
  END;
  SETCURRENT(SL,L(:ASKS:=SL;
  END ASKS;
                                                          END;
```



DOC BUGS CRAWLS OUT :

NOTES ON ALGOL DOCUMENTATION AND BUG REPORTS

D1. — FORE / BACKGROUND [N. Finn]
See N. 3-D18: The method described to find the foreground will work only in a mapped system. The following will work in both mapped and unmapped systems:

```
BOOLEAN FRG;BASED INTEGER IC;POINTER P;
LITERAL USTP(12R8) USTPC(0R8);/*from PARU. SR*/
P:=USTP; FRG:=((USTPC+P)->IC)->IC) AND 1;
```

D2. — FORE / BACKGROUND [C. Waters]
We solved this problem, after much headscratching and reading various DGC manuals, as follows. A procedure INDEV was written, which is called by the main program:

```
EXTERNAL STRING (6) PROCEDURE INDEV;
STRING (6) TTI;INTEGER A,B,C;
TTI:= INDEV(TTI);OPEN(0,TTI);
READ(0,A,B,C);....
```

INDEV is made with the following code:

```
  TITL    INDEV
  .ENT    INDEV
  .EXTU
  .NREL
S=-167
SP=1B0-S
STR1=S
STR2=STR1+2
CSIZE=STR2+2-S
STRFN=7B7+10B11+2
STRPM=2B7+10B11+2
INDEV:    JSR       QSAVE
  CSIZE
  2                 ;2 PARAM.
  SP+STR1           ;INDEV RET ADDR
  STRFN             ;INDEV SPECIFIER
  SP+STR2           ;TTI ADDR
  STRPM             ;TTI SPECIFIER
  LDA     0,STR1,3
  .SYSTM
  .GCIN
  JMP     +1
  LDA     0,STR1+1,3
  LDA     1,CHRCNT
  ADD     1,0
  STA     0,STR+1,3
  JSR     QRETURN
CHRCNT:     6
.END
```

By changing TTI to TTO, and using .GCOUT, a similar procedure for the TTO is made.

D3. — IMBEDDED ASSIGNMENT [N. Finn]
See N3-D1: The operator ":=" has been assigned a lower precedence than "=." Therefore, I:=INDEX(S1,S2)=0 is passed by comparing the result from INDEX with 0, and assigning the resulting boolean to # I.

D4. — FORMAT, OUTPUT [I. S. Wolfe]
My experiments with the built-in FORMAT procedure show the following differences from OUTPUT:
1. Only string and default precision integers can be used.
2. Each variable requires 1 and only 1 "#," and will be output in full. Consecutive "#" 's each represent one variable, printed without intervening spaces.
3. Excess "#" 's, relative to the number of variables, will be printed out literally, i.e. as #'s.
4. One obvious consequence of item 2 is that FORMAT lacks many of the nice features from OUTPUT. [But it has a drastically shorter code!]
5. When one wants to concatenate output in OUTPUT, one can use
OUTPUT(CH, "#<0>#", I, S);
The null byte permits concatenation while preserving all the other formatting power from output.

D5. — UPPER, lower case [I. S. Wolfe]
All Algol users should read the RDOS manual thoroughly. Many of the questions and comments in Nuance are answered there, including LINEREAD's treatment of nulls and linefeeds (like system call. RDL), inability to use lower case on STTI when opened with the standard OPEN, because it does not mask DCLTU, etc. The solution to this problem is to write your own procedure. I call mine "OPN," so I can still use OPEN if I want to. OPN accepts an extra argument, the mask word. If this is the last argument, it can be left out when not wanted and will default to zero. Of course, input of filenames, etc. in lower case is useless unless you write another procedure (mine is called CAP) to shift cases.

continued on page 9

continued from page 8

/**COMMENT:
True, the RDOS and other books should be studied. However, the minimum requirement in the Algol documentation is that it should give references to this kind of information, or better yet, Algol-like definitions of the various procedures. As the example below shows, it is easy to bypass some of the default settings, but no user will know whether this is needed without specific documentation on the built-in procedures.

An alternate, and perhaps better, solution is to maintain the standard OPEN (e.g. with the procedure Console from Algproc), and to write a separate Algol procedure which does a byteread on the console. Then, items which need to be in caps (e.g. for comparision with stored data or sorting) can be read by READ(CNSI,X), typing either upper, lower, or mixed case text. The special procedure (ASKS, see p.7) is used when case preservation is required. As mentioned above, a case shifting procedure also is needed, like the simple one (UPSHIFT, p.5) which shifts to all upper case, or for more fancy shiftwork, CAPSHIFT (p.5) can be used which shift either up, down, or capitalizes words in the first letter only (e.g. the day and month pointers in Algol are returning e.g. MONDAY, whereas Monday is a more useful representation).

D6. — RANDOM [M. B. Ruggera]
The random number generator RANDOM can make negative as well as positive integers: the result is a 16-bit integer with bit 0 as the most significant bit, which is interpreted by regular integers in Algol as the sign.

D7. — LITERALS.
Declaration of named literals in a program is very efficient when the literal does not have to be changed during program run: it can be changed uniformly in the program by an edit and recompilation, without affecting the rest of the code. This is especially true for string literals, e.g. declarations such as TTO("STTO"), AL(". AL"), etc. However, there are problems if very large numbers of literals are used in a program module (see below), and declarations with imbedded brackets are illegal. Use the second statement instead of the first one, substituting octal ASCII values for the brackets. The fact that these octal values are placed within brackets, is the cause for the above requirement.

```
OPER("<>=+-")
OPER("<74><76>=+-")
```

An example of the use of such literals is to extract an operator from an input string by using the INDEX procedure.

D8. — LITERALS [R. Fessenden]
During phase 2 of a compile, the cryptic ERR29 may occur. The book simply says to call DGC. The problem is caused by too many literals in a large, conversational program. Probably an overflow of the literal stack. You can fix it by chaining segments of a program together, or by removing portions of the program to External Procedures.

D9. — LITERALS [M. B. Ruggera]
Using a large number of literals can cause problems (see the letter by Ruggera, this issue). There are several ways to bypass this, using datafiles with the required information. One additional advantage of such data files is that the messages can be modified without having to recompile the program, and that several programs can access the same data. The basic, and simplest, method is to write a string procedure with the following skeleton:

```
STRING PROCEDURE OUTMSG(N);
VALUE N;INTEGER N;
/*Returns message number N*/
BEGIN STRING S;INTEGER I;
FOR I:=1 STEP 1 UNTIL N DO READ(7,S);
OUTMSG:=S;POSITION(7,0);/*reset file*/
END OUTMSG;
```

In the main program, file 7 will have to be opened to the message file, which contains lines with the data. The program uses this procedure as WRITE(TTO, OUTMSG(I)); to type message line I from the file. There are a number of ways to improve the speed: the data can be stored as nodes with the CMM procedures (the first step to a database system!), or (using FILEPOSITION and POSITION) the fileposition of each of the lines can be stored in an array, and then read with only one file access which positions the file to the wanted line, e.g.:

```
POSITION(7,LOC[N]);READ(7,S);
```

The array LOC is filled at the start of the program by reading once through the file.

D10. — OPERATORS
The correct declaration of an operator is

```
INTEGER (2) OPERATOR (|) POWER;
```

But if the precedence specifier (here: |) is omitted, the compiler crashes and results in a TRAP and BREAK, instead of a regular compiler error message.

D11. — WHILE — DO loop [N. Finn]
Although Algol lacks some features of more structured languages, in particular the "repeat until" and the "while do" constructs, the same effect can be achieved with:

```
INTEGER DUMMY:
FOR DUMMY:=1 WHILE ... DO ... ;
```

at a cost of only two useless instructions per iteration (a SUBZL and STA).

D12. — FOR loops [N. Finn]
The syntax of the FOR statement is not clear from the manual. The way to think of it is of the form:

```
FOR <VAR> := <LIST> DO ... ;
```

where <LIST> is a list of elements separated by commas. Each element is one of the forms:
a) <EXPRESSION>
b) <EXPRESSION> WHILE <BOOLEAN EXPR>
c) <EXPR> STEP <EXPR>UNTIL <EXPR>

Form (a) evaluates the expression, assigns it to the variable, and then executes the DO statement once. Form (b) evaluates the expression, assigns it, evaluates the boolean expression, and if true, performs the DO statement. This evaluation, assignment, evaluation, test, perform cycle is repeated until the test shows false. Form (c) is properly explained in the manual.

Thus:

```
FOR I:=1,3 STEP 1 UNTIL 7,I+1 WHILE BOOL DO ..
FOR I:=N,P WHILE BOOL, J STEP K UNTIL M DO ..
```

are legal, but

```
FOR I:=1,3 STEP 1 UNTIL 7 WHILE BOOL DO ..
```

is not: you cannot have two tests ("while" and "until") running at the same time.

D13. — Mixed FOR loops [N. Finn]
Watch out for fancy FOR statements. This one:
FOR I:=5 STEP I UNTIL 7, I+1 WHILE I<11 DO ... is equivalent to
FOR I:=5, 6, 7, 9, 10 DO ... where the "8" is missing, because I is stepped to 8 before testing in the "until" clause, and thus the first "I+1" clause steps it to 9.

/** COMMENT:
For more on loop programming, look back to Celko's article on p.3 of this issue.

D14. — ADDRESS [N. Finn]
The address function is very limited. Use it only for single or subscribed variable names or literals. It fouls up royally on pointer expressions, substrings, etc.

D15. — INTEGER COMPARE [N. Finn]
Signed compares are wrong. The compiler generates code for Nova single instruction signed compares, which are really MOD(32768) compares, not signed compares. The generated code only works when the difference between the compared numbers is equal or less than 32768:

−1	< 0	OK
0	< 30000	OK
−30000	< 0	OK
−30000	> 30000	BAD
−10000	< 22768	OK
−10000	> 22769	BAD

D16. — INTEGER ARITHMETIC [N. Finn]
The following source generates bad code:

```
INTEGER I,J; J:=1; I:=J - J;
```

The compiler generates code for:

```
I := -J -J;
```

D17. — COMPATIBLE FILE FORMAT [J. Celko]
Algol, Fortran, and Basic all can use files having a format with comma separators, quoted strings, etc. Such files cannot be produced with Algol using the OUTPUT or WRITE procedures, because these stuff NULLS all over the file. It is too clumsy to run such files through EDIT or SPEED, or my own NULLKILL program. However, if all output data is changed to strings, the length of these strings can be found, and the appropriate BYTEWRITE used:

```
LITERAL CM(",");INTEGER I,L;
POINTER AS;STRING S;
AS:=ADDRESS(S);I:= ... ;
S:=I CONCAT CM;L:=LENGTH(S);BYTEWRITE(CH,AS,L);
```

after having opened the proper file.

D18. — ALLOCATE [H. W. Eber]

continued on page 10

continued from page 9

If one plans to Byteread a certain number of bytes, and Allocates the exact number of bytes to be read, the Byteread command does not obtain the data:

```
ALLOCATE (P,10);
/*This allocates 20 bytes*/
...
BYTEREAD (CHAN, P, 20);
```

Does not work. This problem again occurs in RDOS Rev6 after having been previously resolved. Some mystified users will be wondering why programs which worked last year, or the year before, no longer do so!

END;

THE COVER

The cover of Nuance 4 is inspired by one of John Cage's musical compositions, which consists of N measures of silence. Nuance's blank space has an even deeper philosophical meaning: it symbolizes the lack of time to prepare the issues, especially in a period where the material available for Nuance is increasing. The editorial on p.2 gives some ideas how readers may help, but any other suggestions are welcome. Perhaps in the next issue there will be quotes again, indicating that there was at least time for reading other material!

END;

A STAIRCASE ??

A program "trick" found in one of the UG Library programs: a staircase function where the height is stepped depending on the riser. To get an average slope differing from 1, a multiplying factor can be used inside the loop.

```
 -6      -8
 -5      -8
 -4      -4
 -3      -4
 -2      -4
 -1      -4
  0       0
  1       0
  2       0
  3       0
  4       4
  5       4
```

```
INTEGER HEIGHT ,RISER,TOP,I;
RISER:=4;
FOR I:=-6 STEP 1 UNTIL 5 DO
BEGIN
HEIGHT:=I AND -RISER;
WRITE<CNSO,I,TB,HEIGHT,CR);
END;
```

END;

LANGUAGES and DIALECTS

A. van Roggen

With the rather hesitant introduction of DG/L on the market, one more Algol-derived computer language has made the scene, among a dozen or so "modern" languages and dialects which are block-structured, procedure oriented, and facilitate both in-source documentation and the building of large, complex program structures.

Anyone who has even casually looked into the plethora of languages, agrees that for the computer architecture of today, the "modern" ones are increasing in usage at the cost of the older ones (Fortran used to be the most used one in this group, although now the number of Basic programmers may even exceed those that work with Cobol). There is also an agreement that all these languages and dialects will last; there are just too many, resulting in excessive maintenance costs, documentation, etc. In the unavoidable shakeout, probably a few will remain which are specialized on applications (e.g. for business, scientific, and control purposes). It is anybody's guess which languages will last; the more machines run a language, the more the durability of that language - witness the tenacity of Fortran.

My guess (prediction?) is that DGC Algol will be completely abandoned by the users and perhaps by DGC, in favor of DG/L, once this much improved superset of Algol becomes available for machines other than the Eclipse. Of course, given enough time, this whole question may become academic: the next "generation" computer architecture will be built to fit the language, and perhaps will be switchable to allow for some degree of multiprocessing or pipelining. A start towards this end already can be seen in the current work on microprocessor networks. Will DG/L stand up against the onslaught of this advance? Based on extrapolation from DGC Algol, the answer is both yes and no. No, because there are not enough users; as a bare minimum it will require DG/L availability not only on Novas, but on microNovas as well, and perhaps on other machines to make it "portable." However, technically, the answer is "yes": the language is better defined than DGC Algol, it allows for easy introduction of new datastructures and can thus be extended by DGC and/or the users. Pointer operators (to these structures) from Algol can be replaced in DG/L by pointer procedures, perhaps with no loss in power. It will be of interest to hear from users who have experimented along these lines.

Meanwhile, for those who want to work with or get a feel for other higher languages besides Algol: both DG/L and PL/I were released by DGC - but for Eclipses only. Nova users can get Pascal (available from Gamma Technology, Palo Alto, CA 94304). This language is also used on microprocessors (e.g. National's) and is strongly Algol oriented, but somewhat less powerful than Algol-60. A very powerful language used for artificial intelligence work, LISP, is also available for the Nova (from the Users Group Library). This language is fully procedure, operator, and list-structure oriented, but does not resemble Algol. Lisp programs see no difference between "data" and "programs" and thus a program can change itself during runtime. This is difficult to accomplish in Algol or DG/L!

END;

Page-fill Procedures

Two useful procedures: a modified Lineread which ends on CR's only and eliminates nulls, and a string compare (upper/lower case) which is alphabetically correct.

```
PROCEDURE READLINE(CH,PTR,CNT,ERLB);
VALUE CH,PTR;INTEGER CH,CNT;POINTER PTR;LABEL ERLB;

/*Fix for DGC's "lineread" to return only lines delimited by CR's. Nulls are discarded.
BEGIN LITERAL NUL("");INTEGER I,L;
STRING (133) S;POINTER P;BASED STRING (133) BS;
P:=ADDRESS(S);PTR->BS:=NUL;CNT:=0;
BIS: LINEREAD(CH,P,L,ERLB);SETCURRENT(S,L);
SUBSTR(PTR->BS,CNT+1,(CNT:=CNT+L)):=S;
IF ASCII(S,L)=0 THEN BEGIN CNT:=CNT-1;GOTO BIS END;
END READLINE;

BOOLEAN PROCEDURE LESSP(A,B);
VALUE A,B;STRING (130) A,B;
/*Returns TRUE when predicate A<B in alphabetic order. Upper and lower case are sorted properly*/
BEGIN
INTEGER L,LA,LB,C,BA. BB:BOOLEAN RES;
C:=0;
L:=IF (LA:=LENGTH(A))<(LB:=LENGTH(B)) THEN LA ELSE LB;
VOLG:
IF C=L THEN
  RES:=IF LA<LB THEN TRUE ELSE FALSE
ELSE
  BEGIN
IF (BA:=ASCII(A,(C:=C+1)))<173R8 AND BA>140R8 THEN
BA:=BA-40R8;
IF (BB:=ASCII(B,C))<173R8 AND BB>140R8 THEN BB:=BB-40R8;
  IF BA=BB THEN GOTO VOLG;
  RES:=BA<BB;
  END;
LESSP:=RES
END LESSP;
```

END;