NUANCE: ℵ2

NEWSLETTER FOR USERS OF ALGOL ON NOVA COMPUTERS AND ECLIPSES

NUANCE 2 1 JULY 1976

ALGOL OVERLAYS

B. M. Friedlander

1.1 Introduction

With the addition of ALGOL0.LB to the ALGOL runtime libraries, DGC Extended ALGOL now has the capability to support RDOS overlays. Through an oversight, the documentation on how to use this capability was omitted. This article explains the usage of the two procedures that handle overlays: OVOPN and OVLOD. OVOPN is used to associate an ALGOL channel with an overlay file. OVLOD is used to load an overlay into an overlay area. To reference OVOPN or OVLOD, they must be declared as EXTERNAL PROCEDUREs. You should be thoroughly familiar with "User Overlays" in Chapter 4 of the RDOS manual (9-75-7 on p. 4-4 to 4-7; in 9-75-6 from p. 4-7 to 4-13.

1.2 Call Syntax

The syntax for the procedure calls are as follows:

```
        OVOPN (<channel>, <filename>);
```

<channel> is the integer ALGOL channel to be associated with the overlay file and to be used by all subsequent calls to OVLOD.

<filename> is a string which contains the name of the overlay file created by RLDR and is associated with the executing program. <filename> may be a string literal, and usually has the extension OL.

```
        OVLOD (<overlayspecifier>);
```

<overlayspecifier> is an integer variable. It must be either an external or a stack integer. Therefore, it must be declared either as EXTERNAL INTEGER or as INTEGER. (regular integers are stack variables). It may not be declared as OWN INTEGER. If it is declared external, then it must be a .ENTO symbol defined in the overlay to be loaded. If it is a stack variable, then it may be either the address of a .ENTO symbol or a numeric value that defines the node/overlay to be loaded and must not be declared as an own integer. The overlay is conditionally loaded; that is, if it is not already in core, then it is read from disk. The normal method of defining a .ENTO symbol is by writing a small assembly language routine that defines the symbol and then loading that routine in the same overlay as the procedure with which it is associated. See "Example Program" below for a demonstration of the usage of assembly language routines.

1.3 Overlay specifiers

The definition of <overlayspecifier> was restricted to external integer and (stack) integer variable to allow OVLOD to recognize .ENTO symbols. Unlike true external integers, the name of the .ENTO symbol is the value of the node/overlay to be loaded instead of the address of the value. This requires careful referencing of .ENTO symbols to pass the correct values to the system. OVLOD may be called with its parameter defined in one of three ways: (1) as a .ENTO symbol, (2) as a stack variable set to the value of a .ENTO symbol, and (3) as a stack variable set to the computed value of the node/overlay to be loaded.

1.3.1 OVLOD (<.ENTO symbol>)

To use a .ENTO symbol directly, you need to use the following constructs. Assume that OVA has been defined by ".ENTO OVA" in an assembly language module. Also, assume that PROC1 is an external procedure that has been loaded as part of the overlay in which OVA is defined. To load and execute PROC1 you would use the following in the main program:

```
     ...
    EXTERNAL INTEGER CVA;
    EXTERNAL PROCEDURE PROC1;
     ...
    OVLOD (OVA);
    PROC1 (...);
     ...
```

1.3.2 OVLOD (<value of .ENTO>)

It might be desirable in a given program to be able to pass the value of the .ENTO symbol to another procedure or to save it for processing return paths. This can be accomplished by setting an integer stack variable to the address of the .ENTO symbol:

```
     ...
    INTEGER OVHOLD;
    EXTERNAL INTEGER OVA;
     ...
    OVHOLD := ADDRESS(OVA);
     ...
```

In this example, the integer stack variable OVHOLD has been set to the value of the .ENTO symbol, OVA. OVHOLD may now be treated like any other

(continued on p. 3)

EDITORIAL - A. van Roggen

The response to Nuance1 has been overwhelming and positive. Of course, with the condition that the mailing list was to consist only of those who have replied, one could not expect anyone to send in an adverse opinion! The number of responses (almost 100) has made the administration a larger task than originally expected; however in so far as it causes progress it is worth the effort. And Algol has made some solid advances lately: overlay documentation, an SOS compiler on the horizon, many bugs were fixed, and at the Users Group Meeting it appeared that DGC management is increasingly interested in providing more users support, including Algol. Let us hope progress will continue at this pace!

We have no doubt that, with all the Algol activity, users will build up a set of general-purpose programs of the kind found on most operating systems (e.g. curve plotting, statistical analysis, sorting, etc.), as well as a set of useful procedures. In this context, we propose that such general programs and procedures as are made available to other users, follow -where possible- a unified form. This proposal is not a great hardship on authors when started now, and will make an Algol system much more useful later. One example is given in ALGLIB for procedure handling, another example for general numerical programs would be to use data input files of uniform format, so that many programs could use the same file without need for reformatting. A similar case can be made for business applications. Standardization is needed for flexibility of operations - this is not a contradiction in terms.

Much discussion will be required to select such uniform formats, for procedures as well as programs. A start can be made with criticism and improvements in ALGLIB. There are other aspects to a unified system. For example, it will never hamper, and even help, development of specialized (e.g. very high speed running) applications. Because the source text is available, specific modifications are readily made, in contrast with canned programs provided by DGC. Another aspect is the distribution of the sources and the required testing. The Users Group Library is being remodeled, and perhaps would be the most logical place for distribution. Testing on several systems should be done on all programs; there are several academic institutions and at least one active high school on the mailing list of Nuance, in which places small projects like these could lead not only to a good educational experience, but to much improved programs as well! (Provided, of course, that they are given not only the well documented source, but also background information, references, and a set of test data). Hopefully, the next Nuance will see a flood of comments on this subject and an avalanche of procedures and programs. This would simultaneously be a positive response to the requests heard during the Users Meeting, asking for more program examples in Nuance.

Finally, the confession that an Editor's life is not all roses. Two of the promised articles for this issue were not received; without contributed articles any newsletter will evaporate. Besides the aforementioned application programs and procedures, there are many other things to discuss: how to handle TTO, TTI on background and foreground; suggestions for compiler improvement (e.g. make 'STEP 1' default in 'FOR I:= 0 UNTIL N'), and how integer arithmetic errors can be circumvented.

NUANCE,

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

Contributors to NUANCE 2 :

A. Burkart
SIEMENS A.G.
Postfach 700076
Muenchen GERMANY

J. Celko
Box 11023
Atlanta
GA 30310

T. R. Churchill
Centre for Min. En. Techn
555 Booth Street
Ottawa K1A 0G1 CANADA

Dr. J. R. Coffman
P. O. Box 2031
Oceanside
CA 92054

J. B. Czulada
The MITRE Corp.
Bedford
MA 01730

D. A. Down
Computer Services Center
Oral Roberts University
Tulsa, Okla 74105

Dr. D. Drew
Computing Sciences
Texas A & M University
College Station TX 77843

B. M. Friedlander
DATA GENERAL CORP.
Rt 9, Bldng 8
Southboro, Mass 01772

R. Gavare
Sprakdata
Norra Allegatan 6
S41301 Goteborg SWEDEN

S. J. Gear
XEROX CORP
800 Philips Rd, 311C
Webster, NY 14580

J. Isaak
DATA GENERAL CORP.
1054 Elwell Court
Palo Alto CA 94303

J. Krichen
Medtronic Inc.
P.O. Box 1453
Minneapolis, Minn 55440

Dr. P. Maas
Univ. of Strathclyde
107 Rottenrow, Glasgow
G4 0NG ENGLAND

M. J. Mardesich
124 NE 58th Street
Seattle
Wash 89103

J. G. Marriott
Parke-Davis Res. Labs
2800 Plymouth Road
Ann Arbor MI 48106

Dr. D. Martin
Dept. Math. Sciences
Memphis State University
Memphis TE 38152

K. M. McClelland
ELECTRODYNE
Providence Highway
Sharon, MA 02067

Dr. C. L. Owings
Computer Engineering
F2705 Mott Child. Hosp.
Ann Arbor Mich 48109

J. Pierce
SW Steel Roll Door Co
9525 White Rock Trail
Dallas TX 75238

J. T. Reynolds
DANRAY, Inc.
13740 Neutron Road
Dallas TX 75240

G. J. Ridsdale
GTE Lenkurt Electric
Burnaby
B.C. V5A 1W3 CANADA

W. D. Selles
NE Medical Center Hosp
171 Harrison Ave
Boston MA 02111

R. E. Williams
Baylor College
Texas Medical Center
Houston TX 77025

B. Wood
Business Directions
Westowne Office Center
Liberty, MO 64068

(ALGOL OVERLAYS cont. from p. 1)

integer and may be passed to any procedure (including OVLOD) as in the following:

```
     ...
    OVLOD(OVHOLD);
     ...
```

This example would cause the loading of the overlay corresponding to the current value of OVHOLD.

1.3.3 OVLOD (<numeric value>)

Nodes and overlays are numbered from zero (0) so that the numeric value for an overlay can be computed by the following formula:

```
<overlayvalue>:=<nodenumber>*256+<overlaynumber>
```

To use this method, you must know into which node of which overlay a procedure is going to be loaded when writing your program. The advantage of this method is that there is no need to write any assembly languages routines to define the .ENTO symbols.

1.4 General Usage of Overlays

Of the three methods described above, the preferable ones are those using .ENTO symbols. Although they require a little additional setup, they make program modification simpler. Using overlay symbols allows you to move procedures from one overlay to another without having to modify the source of the program that does the overlay loading. Using computed numeric values, in this case, would require a change in the loading program.

For ease of programming, each external procedure should be associated with a .ENTO symbol. You may load more than one procedure and its associated .ENTO symbol into an overlay. Using this technique, the sequence of calls for loading and executing procedure "nnn" would be:

```
     ...
    EXTERNAL INTEGER OVnnn;
    EXTERNAL PROCEDURE PRnnn;
     ...
    OVLOD (OVnnn);        /* LOAD PRnnn INTO CORE */
    PRnnn (...);          /* EXECUTE PRnnn */
     ...
```

The overlay load is always conditional so that, if the overlay is already loaded, no load from disk will occur.

1.5 Example Program

This example program consists of four ALGOL modules and three assembly language modules. MAIN.AL is the driver and causes the loading and execution of the overlay procedures. PROC1.AL, PROC2.AL, and PROC3.AL are ALGOL procedures which reside in overlays. OV1.SR, OV2.SR, and OV3.SR are three assembly languages modules that define the .ENTO symbols and are associated with PROC1.AL, PROC2.AL, and PROC3.AL, respectively. The program demonstrates all three methods of using OVLOD.

```
    /*** MAIN.AL ***/
    /*** THIS PROGRAM DEMONSTRATES THE
         USAGE OF OVOPN AND OVLOD ***/
    BEGIN
    EXTERNAL INTEGER OV1,OV2; /* .ENTO SYMBOLS */
    EXTERNAL PROCEDURE PROC1, PROC2, PROC3;
    EXTERNAL PROCEDURE OVOPN, OVLOD;
    INTEGER OVNUM;
    APPEND (0, "$LPT");
    OVOPN(7,"MAIN.OL");/* OPEN OVERLAY FILE */
    OVNUM := ADDRESS(OV1);/*GET VALUE OF OV1*/
    OVLOD(OVNUM); /*LOAD USING STACK VARIABLE*/
    PROC1(OVNUM);
    OVNUM := 0*256 + 1; /* NODE 0, OVERLAY 1 */
    OVLOD(OVNUM); /*LOAD USING COMPUTED VALUE*/
    PROC2(OVNUM);
    OVNUM := ADDRESS(OV2); /*GET VALUE OF OV2*/
    OVLOD(OV2);  /* LOAD USING ENTO SYMBOL */
    PROC3(OVNUM);
    END OF MAIN.AL;
```

```
    /*** OV1.SR ***/
     .TITL     OV1
     .ENTO     OV1
     .END      ; END OF OV1.SR
```

```
    /*** PROC1.AL ***/
PROCEDURE PROC1(OVNUM);
INTEGER OVNUM;
BEGIN
WRITE(0,"HI,I'M PROC1 IN OVERLAY:",
     OVNUM, "<15>");
END OF PROC1.AL;
```

```
    /*** OV2.SR ***/
     .TITL     OV2
     .ENTO     OV2
     .END      ; END OF OV2.SR
```

```
    /*** PROC2.AL ***/
PROCEDURE PROC2 (OVNUM);
INTEGER OVNUM;
BEGIN
WRITE(0, "HI, I'M PROC2 IN OVERLAY:",
     OVNUM, "<15>");
END OF PROC2.AL;
```

```
    /***OV3.SR***/
     .TITL OV3
     .ENTO OV3
     .END      ; END OF OV3.SR
```

```
    /*** PROC3.AL ***/
PROCEDURE PROC3 (OVNUM);
INTEGER OVNUM;
BEGIN
 WRITE(0, "HI, I'M PROC3 IN OVERLAY:",
      OVNUM, "<15>");
 END OF PROC3.AL;
```

```
     /*** CLI COMMAND TO LOAD THE PROGRAM ***/
     /***AFTER FIRST COMPILING ALL AL AND SR
          MODULES***/
RLDR/A/E/P $LPT/L MAIN [OV1 PROC1,OV2 PROC2 ↑
     OV3 PROC3] @LIBRARY.CM@
```

The result of executing this program is:

```
     HI, I'M PROC1 IN OVERLAY:0
     HI, I'M PROC2 IN OVERLAY:1
     HI, I'M PROC3 IN OVERLAY:1
```

1.6 Conclusion

Overlays provide the capability for folding large ALGOL programs into a small address space. I hope the information contained in this article will be of help in using ALGOL overlays.

END; END;

A. van Roggen:

CALL ALEX,

to cut the Gordian Knot

Ah, but a man's reach should exceed his grasp
Or what's a heaven for ?

R. Browning

Alexander the Great, after conquering Phrygia (central Turkey), and in a push to capture the rest of his empire, had no time for untangling the famous farmer's knot, and cut it with his sword. Indeed, why should one worry about untangling knotty problems in compiling, assembling, loading, and executing Algol programs, when ALEX (Algol EXecute) can do the job faster and with fewer errors, and leaves time for the programmer to do more important work.

Before any Algol program can be executed, it must be edited into the computer and compiled. It must then be loaded, usually with INCLUDE files and with separate Algol and ASM procedures (each of which has to be edited and compiled), and with the appropriate Algol library routines. Occasionally, the programmer may have made a (typing) error, or in other (equally rare) cases the documentation is ambiguous. When this happens, further editing is required in one or more of the modules, followed each time by recompilation of the updated modules and the inevitable reloading, whereby the operator must keep track of the proper names of all the arguments and their order. All this is accompanied by laborious typing dc CLI commands. A short cut can be made by using indirect files that contain the needed CLI commands. However, only the edited modules have to be recompiled, and indirect files are thus wasteful of computer time.

ALEX acts like a CLI command, somewhat similar to CLG for Fortran, but on a level of sophistication that rivals large commercial timeshare systems. The arguments for ALEX are the main program name and the separately compiled modules. For example, ALEX TESTPROG PROC1/S would execute TESTPROG with a separate ASM procedure PROC1. With these two commands, ALEX first types the time and date and main program name (TESTPROG). It checks the presence on disk of TESTPROG.AL, TESTPROG.DC, and TESTPROG.RB. If the RB file does not exist, or if its creation time is earlier than either or both the AL and DC files (indicating a recent update to these files), TESTPROG is recompiled, i.e. the CLI command ALGOL TESTPROG is executed. If there was no update, this compilation is skipped.

The DC file, if used, contains the program comments, operating instructions, and the declarations for the program, see DEMO below. This file is INCLUDEd in the main program; it is very convenient for editing during main program development. After the program is 'finished', the DC file can be edited into the main program instead of the INCLUDE statement.

The next step in ALEX is a similar checking, and compiling when necessary, of each of the following arguments, here PROC1. The local switch /S indicates that it is a ASM source and that the time of PROC1.RB has to be compared against that of PROC1.SR, not the AL file. After the procedure modules have been taken care of, the loader is called, and the correct CLI command for the RLDR is given, but again, only when any of the RB files encountered is more recent than the TESTPROG.SV file. Finally, the CLI command 'TESTPROG' is given and the program is executed, after which the CLI is called back. The ALEX command, as shown above is an enormous condensation of operator action required to build a working Algol program. However, it can still be shorter! For the example shown, if the results are not quite what was expected, editing would follow, e.g. on PROC1.SR. Then the simple command 'ALEX' would repeat the previous ALEX command, without having to repeat its arguments (these are stored in a small ALEX.CM file). This command would cause only compilation of the procedure, reloading, and execution of TESTPROG. Perhaps this is the ultimate for slow typists. For programmers who would want to worry more about programs than about compilers and loaders, it is so automatic and convenient that one person (who will remain nameless) was known to have forgotten the main program's name while testing a procedure! Now ALEX types both the time and the main name before running. More visual aid can come with the global switch /V which verifies on the TTY what action will be taken (e.g. ASM PROC1, etc.) before ALEX starts such action: a quick control-A will abort the operations.

Other global switches are /L which causes the listing files to be kept, including the load map, and /N which omits the final execution of the program, i.e. stops after the RLDR command; this feature is used for programs that require arguments (like ALEX) or for those that may abort during execution and require operator attention. The only other switch is a local one, /D to be used for library files and in case an RB module should not be recompiled.

The method used by ALEX is to check its arguments first, and to create a command file ALEX.CM (similar to COM.CM) or FALEX.CM in the foreground. In case Alex was called without arguments, the command files are renamed, and operation continues with the arguments found in the renamed file. From the files found on disk and their creation time (from the file UFD, see RDOS), ALEX generates the proper CLI commands and dispatches them to a file CLI.CM, and finally chains to CLI.SV which executes the CLI.CM commands. This, perhaps, is a weakness of ALEX. Because it uses CLI.CM rather than COM.CM itself, it can be used only on RDOS systems from rev 4.00 up. On the other hand, were COM.CM used, one more level of push depth would be required in the program, and there are not many levels as is! In the example below, editing was done with SPEED, and in one case no changes were actually made in DEMPROC.AL to show the shortened operations from ALEX.

```
TYPE DEMO.AL
BEGIN /*DEMO.AL*/     Main program
INCLUDE DEMO.DC;
CONSOLE;
WRITE(CNSO,"IN DEMO ");DEMPROC;
END;
R
TYPE DEMO.DC        Separate declarations
/*NO OPERATING INSTRUCTIONS NEEDED*/
LITERAL CR("<15>"),SP(" ");
INTEGER I;REAL R;STRING S;
EXTERNAL INTEGER CNSO;
EXTERNAL PROCEDURE CONSOLE,DEMPROC;
/*NO LOCAL PROCEDURES*/
R
TYPE DEMPROC.AL
/***DEMPROC***/     External prócedure
PROCEDURE DEMPROC(S);STRING S;
BEGIN
LITERAL NUL(""),CR("<15>");EXTERNAL INTEGER CNSO;
IF S=NUL THEN WRITE(CNSO,"NO ARGS",CR)
  ELSE WRITE(CNSO,"DEMPROC S= ",S,CR);
```

Continued on page 11

Letters:

/** COMMENT: Many more comments and letters have been received than can be reproduced here; those of general user interest are partly excerpted here, some are reformatted into the Q/A section or into the D(ocumentation) section, and in a few cases, cross references have been made.
; END **/

FROM: B. Wood, Business Directions.

Business Directions operates a data processing service, utilizing a NOVA2/10 for financial and general business operations. Presently, most of our programs are written in Fortran, a package acquired from The Automated Quill in Denver.

Since it is obvious that Algol is much more suited to the business environment, I have been attempting to convert some of our more frequently used routines to Algol coding. I have found, however, that my version of Algol is full of bugs. This version is dated 1973, and I would assume that DGC has cleared up a large percentage of these bugs since that time.

I would appreciate your evaluation and comments concerning the most recent versions of Algol to help me determine whether or not to acquire the last version and to continue the conversion.

/** COMMENT: Rev 2.03 of Algol has many bugs cleared up, although it is not yet out of the woods. Many of the problems remaining (such as integer arithmetic) may pose difficulties in scientific applications, but would not be noticed in business and systems programming. For me, the software subscription for Algol (numbers 3222 and 3218) was worth the money, and more fixes are on their way.
; END **/

FROM: J. Celko, Atlanta GA

For the past few years, I have been writing business applications at turn-key houses for Novas - in Fortran, with the commercial subroutine package or a local version of it. This is generally done because (1) the boss thinks it is easy to get Fortran programmers, (2) they do some work for an 1130 and feel good about Fortran, (3) everyone else does, and (4) they feel they need multitasking.

However, I hate this because (1) I'm an Algol freak and get sick at the sight of Fortran, (2) the "faking" that has to be done to get long integers in Fortran cost quite a bit in terms of documentation and computer time - every operation is a subroutine call, (3) I am seeing more and more people who know Algol, or an Algol based language of sorts - structured programming seems to be the reason for this, and (4) multitasking in a commercial environment generally means multiple CRT units or simple foreground and background work.

/** COMMENT: Joe also has contributed to the Cover and the Q/A section. I have received many comments on the suitability of Algol for the business world (this was news to me!) not only because of the adjustable precision, but also due to the string operations, the facility with which e.g. filenames can be calculated in programs, etc. Algol now has overlay capability, and perhaps multitasking will follow. Let us keep hoping.
; END **/

FROM: M. J. Mardesich, Seattle WA

My first observation on DGC Algol concerns a restriction which is not sufficiently documented. This will be of particular interest to users who contemplate writing LL(1) syntax analyzers using recursive decent. This restriction voids global variable access from an environment whose current activation history contains block levels which are higher than itself. For example:

```
BEGIN
INTEGER A;...
PROCEDURE PA;
  BEGIN ... <ref to A> ... END PA;
PROCEDURE PB;
  BEGIN
  PROCEDURE PC;
    BEGIN ... PA;...  END PC;
   ...    PC;   ...
  END PB;
 ... PB; ...
END OF PROGRAM;
```

In this program the reference to A from PA is never realized and frequently no error indication is given; just erroneous results. A workable solution is to declare variables with such access OWN, but not of course, when the variable has to be dynamically allocated!

Point 2: A forward procedure declaration capability (e.g. such as in Pascal or Burroughs Algol) would be useful for highly recursive algorithms such as LL(1) parsers. The current work-around is to use a switch rather than separate procedures, which results in a reduction in clarity:

```
PROCEDURE PARSE(WHAT); INTEGER WHAT;
  BEGIN ... GOTO ANALYZE[WHAT]; ... END;
```

which leads me to ask for a nice CASE statement such as in most Algol implementations.

Finally, a good formal syntax definition of DGC Algol (which must exist somewhere at the factory) would be highly useful for programmers new to DGC Algol who are familiar with formal definition notations. This, in fact, will be usually the case since most new DGC Algol users will be previously exposed to other Algol or Pascal or PL/I. The ad-hoc Fortran style prose is just not up to the traditional Algol documentation quality.

Barring the above, DGC Algol has shown itself to be a most satisfactory tool.

/** COMMENT: I am not optimistic that such a definition exists, by reason that an Algol style definition works as a systemic insecticide: it pervasively penetrates the language, kills bugs throughout and, like systemic insecticides, not instantaneously! However, the point is well taken and an effort in this direction might not only

help DGC in a cheaper bug killer effort, but will save the users much grief (and time and money) by not having to guess at the definitions.
; END **/

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

We have a small shop engaged in biomedical image processing research. We run a 128k Nova 840 with 232M bytes disk storage. This system supports two custom designed image scanners, a flying spot film scanner and a television microscope.

We mostly use Fortran4 but regard Algol as a better implementation. We have used Algol for work out of the mainstream of our program development, notably for an assembler for a microprocessor.

We hope to be able to contribute to Nuance.
; END **/

FROM: K. M. McClelland, Electrodyne.

Thanks for a Nuance that is not subtle! I am sure that all who share my love/hate relationship with DGC's whimsical compiler welcome Nuance. I wish you luck in handling the job.

One thing Nuance should do is to announce revisions to run time libraries and compilers (RDOS, RTOS, or SOS versions). Until recently I was working with quite an old revision and did not realize there had been improvements. The CLI command REV makes it easy to keep track of what you have. It would probably be wise for everyone to mention the rev he uses in questions and discussions. Perhaps someone from DGC could give a synopsis of the various revs they have sprung on us over the years.

Nuance1 was an eye-opener for me. Nowhere in the manual did I ever see any reference to the PL/I style /*comments*/. Every time I used COMMENT ...; I cursed the lack of the PL/I form, never dreaming the lack was confined to the documentation.

For my applications, the greatest failure of the DGC Algol compiler is its inefficiency in coding. I am primarily annoyed by the compiler's insistence on making a boolean variable out of an integer expression, and testing that rather than simply testing the integers. Similar poor implementation prompts the compiler to do LDA, NEG, ADD when subtraction of a constant is required.

The former problem could be partially allayed by requiring that Algol use MAC rather than ASM, and providing smart macros. In fact, it would not be difficult to design a single pass program interposed between Algol and MAC which would tell the macros when they could use direct jumps. I have found in my work that MAC can take a great load off an Algol program which prepares assembly language files.

My major use of Algol has been a systems generation program that makes SYSGEN look simple. It accepts TTY or file input, performs an operation on it that is equivalent to compilation, produces several intermediate files and listings sorted in various ways, chains down to MAC to assemble two different files of tables (much of the intelligence in the file creation process resides in the macros), then down to RLDR to load the tables along with a library and some loose files, and finally down to CLI.SV to make a self-loading paper tape image. All this is designed to be run by people who have not the foggiest notion of what is going on. Such a program would like about 30k of user space. Not being so blessed, I use overlays.

Although I admit I don't know anything about DGC's OVLOD, I have developed a relatively simple scheme for using overlays. Each overlay segment is a procedure. In this scheme, first an OVOPEN must be performed. Then, when a procedure in an overlay (say in node 2) is to be called, the following statements are written:

```
NODE2:=<literal which is node and overlay number>;
CALL2  /*whatever calling sequence the overlay
 requires*/;
```

Both NODE2 and CALL2 are declared external. CALL2 does a juggle with SAVE and .OVLOD 77, with the result that the called procedure thinks it was called through a JSR @CALL. Returns happen normally.

I have also made a version of CHAIN which allows use of the full capabilities of system call .EXEC (e.g. chaining to another level and using the Debug starting address).

If anyone would like more information on these routines or source tapes, I will be happy to provide them.

/*COMMENT:
Starting with your last offer first, what about writing up the procedures in more detail for the next Nuance, so that "takers" can judge more carefully whether they could use the programs? This might save some work also in tape reproduction. The Users Group Library would like to have copies of your program. Your overlay scheme looks somewhat similar to the "official" one; you must have figured it out the hard way. How many hours would you have saved, had the documentation existed? I wholeheartedly agree with the necessity of mentioning the REV numbers in correspondence. Some of the bugs shipped to me here obviously petrified ones out of DGC's prehistory; of others I am not sure, and from the amount I received and the lack of specifics, I cannot possibly try to see whether they are still crawling around. Some of your items have wound up in the Q/A and D sections.
; END **/

FROM: C. L. Owings, Mott Childrens Hospital.

I am currently working on a modification of the Algol compiler to allow it to run under SOS. Since this is a project which has been pushed aside as more urgent problems are handled, it will probably not be finished until late this summer.

/*COMMENT:
Those of us who heard Professor Owings' talk on computerized measurements on people, during the Users Meeting in May, will recognize that an SOS compiler is of great importance to his work, where the measuring computer is more ambulant than some of the measured people, and a disk system would be awkward if not unreliable. The same holds for many other practical applications where microcomputers (microNovas) will now be used. We wish him luck with the modifications, and look forward to seeing this program in the Users Group Library, where it would be contributed and available to all SOS-ers. At "press" time, word is that DGC will provide Dr. Owings with a current version of the RDOS compiler which has the old bugs of the paper tape system removed; the SOS system will thus be equally reliable as the RDOS system. This is really good news!
; END **/

FROM: J. Isaak, DGC Palo Alto

I am an avid Algol user, and I have done a number of applcation programs in Algol prior to working for DGC. In addition, I taught a class in Algol for the Bay Area Nova Group.

My two concerns at present are: (1) a lack of Algol support for DOS systems, and (2) getting enough company names (with $$ value of equipment) that use Algol to backup the contention that Algol support is a significant DGC product.

/*COMMENT:
Perhaps Nuance can help, indirectly, in answering your second point. Original estimates were that at most 10 responses would be received on Nuance1. This, it turned out, was about one order of magnitude underestimated [due to the number I have no exact count yet; I am still working on the addressing system. In this respect, it is more than I had hoped (or bargained!) for, but the administration will straighten out gradually]. As a matter of fact, the Users Meeting session for Algol specialists had about 10 people (until the competition with RDOS, which was run in parallel), and an almost full room for the "general" Algol session. Maybe someone from DGC took accurate counts. Regarding your first point, I fully agree that the compiler should be made available on levels "lower" than RDOS; Dr. Owings' letter (above) and others (see Q/A 4 and 5) indicate active interest. I am in the process of getting a microNova system, which I will run in Algol, although it must be compiled on the disk. I expect this will be a nuisance, but I will find out experimentally.
; END */

FROM: S. J. Gear, Xerox Corp.

I have read the first issue of Nuance and would like to add my three cheers. We have used Algol here for about 2 years. It is a very fast way to implement programs for text manipulation and even for file manipulation. We have also written several versions of a "TEST LANGUAGE COMPILER" in Algol. These are extremely large programs that won't fit in our 32k Nova2, so we have implemented an overlay procedure, which, although cumbersome, has worked. We have struggled through many cans of RAID and have even forgotten many of the species put to rest. Certainly Nuance would have saved us, and definitely will save new users, much head scratching. [Not on the disk, I hope. Ed.] It provides a central place to document the problems and solutions.

I have done extensive recoding of the assembler code produced by Algol - to reduce program (overlay) size and improve run time efficiency, and of course, to remove bugs. We may be able to answer some future questions when they ring a bell. However, we are still neophytes, and Nuance will provide us with a wealth of knowledge about the mysteries of DGC Algol.

/*COMMENT:
Perhaps the overlay article in this issue will help to smooth these wrinkles in your programs. I believe there is considerable interest in compiler writing among the Algol users. Could you contribute an article on the approach you took and illustrate that with excerpts from your programs? It will save other Algolisti lots of work when they read about your experiences.
; END */

FROM: J. G. Marriott, Parke-Davis Res. Labs

When we started our system, I didn't even get to load the Algol tapes. We were just interested in getting running, not trying new languages. Now that I have had a chance to get a look at Algol - primarily through Nuance - I am looking through the cupboards to see where we stored all of those Algol tapes. It looks verrrrry interesting.

I would like to be put on the mailing list of interested Algolisti. Although it may take some time to get acquainted with a new language, it definitely looks like it may be worth the effort.

/*COMMENT:
There are quite a few newcomers to Algol, both to the DGC version, and to Algol "an Sich". Especially for the latter group, it may be worth while to show program examples. Such a request was also made during the Algol-SIG meeting in Lake Geneva. To make this effective, write for requests: that will help in making a balanced menu for future issues. To some extent, small examples are already shown, and where possible with cross references, but it is not a systematic effort. Those who have found interesting or enlightening ways of solving program problems, are encouraged to write the solutions down for inclusion in Nuance.
; END */

FROM: J. Krichen, Medtronic.

Because of the string handling capability, I am very much interested in Algol, and would like to use it in simple database applications. I am working on a key word out of context filing system in Algol.
; END */

FROM: G. J. Risdale, GTE Lenkurt Electric.

We use Algol quite heavily for string and list processing procedures, primarily on a NOVA 800. One comment: Although the TI700 is a fine interactive terminal (we share your carriage return problem) it does make for difficult reading, when reproduced. Still, it was worth the effort!

/*COMMENT:
For the answer to the TI700 return, see the Q/A section.
; END */

FROM: P. Maas, Univ. of Strathclyde, (G.B.)

Congratulations on a first class start with an excellent and needed idea. Please include me on the mailing list; you have done us all a great service.

I have some comments on the Algol I/O, which might be a bug (?). All procedures, other than BYTEWRITE, use nonprinting separators (chiefly nulls) between values output to files or devices. This is incompatible with reading such files with Fortran, Basic, or as input to the Assemblers. The only way I have found around this problem is to flush all such files through EDIT which deletes nulls. This is not really satisfactory in a multi-language software system: we don't fancy translating all Fortran software to Algol or the reverse. It would seem that these procedures should use printing separators rather than nulls. Has anyone found a better solution?

I also echo the request for multitasking in Algol, together with access to system calls (Why provide CHAIN, and not SWAP?). Lastly, since DGC Algol cannot process call-by-name, but only call-by-reference, could they not arrange to handle Fortran subroutine linkage with the same routines as Algol, thus permitting limited mixed language programming?

/*COMMENT:
I agree with your complaint about the non-printing separators. It must have been an unprintable character who invented this devilish device. Try to make fixed length records, e.g. for Cobol or RDOSSORT with "STRING (50) S; ... WRITE(CH,S);" and behold, RDOSSORT needs 51 byte long records to read properly. I wonder how many have fallen in that trap, as I did. And only a byte analysis of the file will show this; analysis and a trial to correct with EDIT changes the required length to 50, even without doing "anything"! Extremely confusing, and not well documented.

A similar problem occurs in a formatting program I use for documentation of programs. The Algol compiler listing prints innocently looking lines such as "1 BEGIN", but which contain one or more nulls (e.g. after the 1). A READ(CH,LIGNE,EOF) cannot be used, because some lines accidentally start with a double quote ("), which is interpreted as the delimiter. But with a LINEREAD two separate lines are read in the example. The only way I found around this was to use an extra string:

```
INTEGER L,C;STRING LIGNE,ENCORE;POINTER PTE;...
PTE:=ADDRESS(ENCORE);
NWL: SETCURRENT(LIGNE,0);
NWLS: LINEREAD(CH,PTE,C,EOF);SETCURRENT(ENCORE,C);
L:=LENGTH(LIGNE);SUBSTR(LIGNE,L+1,L-1+C):=ENCORE;
IF ASCII(ENCORE,C)=0 THEN GOTO NWLS;
CONT: /*here input, terminated with CR and not
with null, is found in LIGNE*/
```

Rather roundabouts, but it works (so far, anyway).

Access to systems calls now can be done as shown in Nuance1, but there is a better method on the horizon. One accomplished and heavy user of Algol has a "universal" systems call procedure, which also returns the AC's, e.g. for error processing. I understand that this will get into the Users Library and am looking forward to using it! This will shorten my library procedures considerably.
; END **/

FROM: R. Gavare, Sprakdata (Sweden).

In the Department of computational linguistics at the University of Goteborg, we use Algol on our NOVA 840 for research work - for batch processing, timeshare is done in Extended Basic. For your information, a research report is enclosed.

There are some problems with the Algol, and I have several comments. The conversion from multi precision integers to real numbers does not work, and conversion of precision within arithmetic expressions should be discussed in the manual. Why does the multiprecision relational operation give 0 errors in the assembler? Finally, the Procedure list (Nuance1, p.11) lists REM as needing external (I think it is not required) and does not show the identifier FORMAT, which is not in the manual either!

/*COMMENT:
This research is impressive indeed, from the point of view of dataprocessing as well as for linguistics; it is not limited to dictionaries, but also concerns itself with the scientific analysis of language. I appreciate reading about this. I have received many letters, in all forms, from handwritten notes, through typed memos to lineprinter sheets; perhaps some day a palimpsest will arrive. (For those who don't ever use foreign languages, including Algol, a palimpsest is not a message written in Pali). Some of the bugs you found are in the DOC section. The problems with consoles using lower case are manifold, in some cases, strings in Algol are translated to upper case (I have not traced this yet, perhaps someone knows?). To avoid difficulties, I use upper case for all Algol commands, procedures, and use lower case only in the printed literals and strings.

You and your colleagues must have built a large number of string procedures for the work at Sprakdata. Could you write about these for a future issue of Nuance?
; END **/

FROM: D. A. Down, Oral Roberts University.

I congratulate you on your efforts to organize us isolated Algolisti in our struggle with the mysterious quirks of DGC Algol. I have frequently had my enthusiasm for Algol dampened by whimsical bugs that pop into existence in the midst of the purest source code. I have concluded that I'd better not venture into any new features with application programs until I've conducted a thorough investigation for bugs. I've found the notes in Nuance enlightening, and hope that DGC takes appropriate corrective action.

Some of my experiences are:
EXTERNAL VARIABLES. These reside in page zero and quickly use up what little space is available, while requiring an ASM routine to define them. Seldom worth the extra effort required for use.
LITERALS: Beware of excessive number of literals. If more than 256 words are needed on the literal stack, the compiler will output a cryptic message and expire.
PROGRAM TITLES: If you want a title on the main program RB, or need an extra symbol for the beginning of the program, put a label on the initial 'begin', i.e: "NAME: BEGIN ...".
FORMAT: A useful procedure similar to OUTPUT, with the following difference: each occurrence of "#" constitutes a separate field specifier.

/*COMMENT:
Other items are found in the DOC section.
; END **/

FROM: D. Martin, Memphis State University.

I have just managed to get a copy of Nuance1, and would like to be put on the mailing list. We use Algol exclusively in a course in compiler design where recursion was very helpful. Fortunately, the problems we attacked were "well formed" for recursive solutions so we did not build up long run times as you undoubtedly did in the recursive solution to the Fibonacci numbers. I might note that recursion is very helpful in many non-numeric problems and that Snobol does allow recursion. Snobol is certainly not a "simple, straight through" language.

/** COMMENT:
I know. It serves me right to get wrapped on the pen for expressing myself poorly. In this issue, I have made apologies to Snobol. Could you or your students contribute by showing examples of the recursion you descibed, and the procedures used? Do you know of a more precise way of measuring runtime of procedures: the one shown after the Recursion article (this issue) is not very satisfactory for short procedures.
; END **/

FROM: D. Drew, Texas A&M University.

A copy of Nuance1 drifted in from some unknown source two days ago. Since we have several classes that are required to use DGC Algol, we would very much like to continue receiving the newsletter. We should be able to make some contributions in the form of honor stories, but I do not have anything written yet.

We have an Amdahl with IBM Virtual, Hasp, MVT, and so on that does the majority of our computer processing, but we are building up a Nova Lab also to show our students the difference between luxury and economy models. We have a lot of complaints about DGC Algol. I wonder if anyone knows how many ways the compiler can die without giving any hint that anything is wrong?

If you like foreign proverbs, perhaps you will like the old Spanish proverb that describes anyone who can use DGC Algol even a little:

En la tierra de los ciegos el tuerto es rey

/◆◆ COMMENT:
How true. When I use this Algol, sometimes I feel like the above Spanish Cyclops after he lost his one contact lens. In relation to your question on the compiler, there is absolutely no relation between the answer and the present cover design (I hope). More seriously, be sure to use the last Algol release (Rev 2.03) to eliminate many of the older bugs. Captain Hopper (who spoke at the Users Meeting) would take argument with you on the remark of the economy models: she predicted that the big ones would go the way the big dinosaurs went, and be replaced with minis linked together.

I hope you will seriously consider letting some of the students work on a problem, using DGC Algol. It will be useful for them and for the other users as well. Some suggestions are in this issue's Editorial. Let me know if you need the source of the various procedures or programs or if Nuance can be of more specific help.
; END ◆◆/

FROM: J. T. Reynolds, Danray Inc.

While we at Danray do not use Algol for our production work, I am a strong advocate of its use where possible. To this end I am interested in techniques to write efficient code in Algol, optimize its source code, and optimize post-compilation ASM code. Could you put me in contact with others who have an interest in this area?

/◆◆ COMMENT:
I found it useful to write "standard" Algol with indices etc. and make the program work, and then to streamline the code by using pointers and eliminating the index calculations. I plan to use a matrix inversion procedure as example, perhaps in the next issue. Using pointers for integers does not help the running speed. Once this works, you can optimize the ASM code by doing ALGOL/S which produces the ASM text, and make shortcuts there. Minor ones can be made rather easily, but one of the great unknowns is the time spent in the built-in procedures. To my knowledge, there are neither documentation nor flowcharts for these.
; END ◆◆/

FROM: J. R. Coffman, Consultant, Oceanside CA

I read with interest the first issue of Nuance, and hope that the attempts to get an Algol users' forum into operation prove successful. I believe that DGC Algol is potentially the most useful language under RDOS, but that the compiler, at least under RDOS 3.01, is in need of serious debugging. We have just received RDOS 5.00, which I hope, has a new Algol compiler, but I have not yet had the chance to try it.

Our Algol use has been largely incidental; that is, I have used it for writing utility programs which had to do text manipulation, and somehow have been able to avoid the worst of the compiler bugs. We are currently running an improved version of the DGC Fortran IV post processor which converts more subscript references to in-line coding than did the DGC version, fixes numerous bugs, and also converts multiply code to in-line code. I mention it because it is written in Algol. Likewise, our RUNOFF program, which formats documents for publication. I gather it is similar to what was used to produce Nuance.

By way of contribution to Nuance, I am enclosing the description of our approach to the TI Silent 700 problem (Nuance1, Q/A 6). I opted for a hardware fix in the terminal because I felt that the slow carriage return was a hardware problem, and I was too lazy to take on the task of forever modifying DGC software. This solution has been provided for a number of other users in the San Diego area.

/◆◆ COMMENT:
I had the pleasure of meeting Dr. Coffman in the past on another subject; as I would expect, he has indeed the correct approach by fixing the hardware. From another source, I got another hardware solution, which puts a delay circuit on the TTY board in the Nova, instead of modifying the terminal. Both methods work fine, and both leave the option to use either the computer with another terminal, or the terminal with another computer. I checked with the DGC Users Group Hardware Chief (H. Pottinger), and he has records of the solutions, in case others want to use these terminals.
; END ◆◆/

FROM: J. B. Czulada, The Mitre Corporation.

I have had moderate experience with DGC Algol over the past three years, with my own programming amounting to approximately 10000 lines of Algol code written for the Nova 800. During that time, our programming group has encountered numerous mysterious problems. We have found that we cannot trust global variables after program flow exits from a FOR loop. In addition, exit from two nested FOR loops causes a systems crash if their END statements are adjacent. The latter problem is particularly annoying to structured programming types.

We assume that the trouble rests with faulty reinitialization of the stack pointer upon exit from loops. However, we haven't chased the problem down yet. Also, we have endless troubles with both integer and real arrays. We even have occasional problems with the SUBSTR function not

Continued on page 11

NOTE: "A Primer of ALGOL60 Programming" by E. W. Dijkstra, reviewed in Nuance1, is on special sale at half price from Academic Press.

A. van Roggen:

ALGLIB

OR

TOWARDS A UNIFORMLY DOCUMENTED ALGOL SYSTEM

He that will not apply new remedies must expect new evils;
for time is the greatest innovator.
Francis Bacon

The title almost read "universal", but that would have been overly optimistic. Not that a universal computer language and documentation system would not be welcome, or even a small step towards one, but Algol60 (or the DGC Algol) would never make it. Algol68 would have a much better chance, and by now there are perhaps still better candidates. Nevertheless, uniform documentation is a lofty goal: 'T is a consummation devoutly to be wished, as the Danes are quoted to say. Certainly, documentation is essential in any system, and good and uniform documentation in individual users' systems is a necessary first step in getting programs "portable" if not understandable and operable. It also may be a reminder for manufacturers to provide and use better documentation!

Algol is an easy language for documentation purposes; the nature of the language is such that it is almost self documenting: compare reading an Algol procedure with a similar one in Fortran or Basic. Furthermore, additional comments can be added easily, even inside statements, while the free format and the inherent block structure can make the program even more clear. The library procedures always require ample documentation. Some modules are used only sparingly and may require scrutinizing before being used in a new surrounding, or certainly after they have produced unexpected results! It is thus essential to have the source code, either Algol or ASM, readily available, and there should be a guarantee that the current text indeed corresponds to the current library relocatable binary.

On the DGC machines, the library RB's are combined in a file with LB extension, which is invoked with the loader command (RLDR). Upkeep of the LB file is easy with the library file editor (LFE), which extracts, inserts, etc. the various modules. However, after doing this on a procedure PR1.AL, the system has on disk PR1.AL, PR1.RB, and LIB.LB. The RB file is superfluous and should be deleted; the LB file is used most, and the AL file has the text reference. With many such procedures, the directory is getting clogged with names. Keeping the records straight is a major task, not to mention the chance of accidental similarity in filenames. A simple solution to this problem is the use of two extra files, LIB.LF and LIB.LT. Before being deleted, PR1.AL is appended to LIB.LF which contains the current text records while the LT file has all the module names (here: PR1).

Of course, having a systematic way of treating the library modules, it follows immediately that this now should be done by the computer, relieving the operator from enormous amounts of tedium and possible errors associated with the individual steps in the upkeep. This automation is just as important for a smooth operating system as proper documentation, and still leaves a large amount of flexibility for "individuality" in systems: it is applicable wherever Algol is used best, in scientific work, business administration, string and database manipulations, and in interactive processing. A short description of such an automated system follows.

The files associated with the system are ALGOL.LB, ALGPROC.LB, ALGPROC.LF, ALGPROC.LT, and ALGLIB.SV. ALGOL.LB is merged with LFE from the DGC library files ALGOL0,... and the multiply/divide tape (see Nuance1, 0/A1). This allows easy updating when new versions are issued by DGC. Library procedures from the Users Group, or homemade procedures form the ALGPROC files, which will be abbreviated (in the text only) as LB, LF, and LT. The file ALGLIB.SV does all the manipulations on the library. Most straightforward are the LB file, formed with the LFE commands, and the LT file which is just the "LFE T" version of the LB file. The LF file, as mentioned, has the texts.

In order to facilitate the handling and provide clearer printout of this file, the modules are marked with begin and end strings (similar to the begin and end blocks in the LB files). The end string is a blank line, followed by formfeed, <15><14>, so that there is separation for printing on terminals, and a page throw on lineprinters. The string demarcation at the beginning of the module is /***XYZ***/<15> for a procedure XYZ.AL, or ; /***FOR***/ for procedure FOR.SR. This string is an Algol comment (and/or ASM comment) and serves as a clearly visible header. See the procedure FIBONACCI elsewhere in this issue.

The operation of ALGLIB is consistent with that of LFE: it is called in the same way and with similar arguments. For example, INIT.SR is a procedure to initialize devices, and thus is in ASM, while FIBO.AL is an integer procedure in Algol. Assume that FIBO already is in the library, but must be updated, and that INIT is new. This procedure has been tested separately as external procedure to some calling program, and thus both the SR and RB files exist. Assume further that INIT must be inserted before FIBO (Reasons for this are that FIBO uses INIT as a declared external procedure; see the LFE book for more details). The CLI commands to update the library are:

```
   ALGLIB I FIBO/B INIT/S
   ALGLIB X FIBO
   SPEED FIBO.AL;ALGOL FIBO
   ALGLIB R FIBO
   DELETE INIT.- FIBO.-
```

(No details shown for edit, compile.)

The extract function, X, makes a new source file only, and does not extract the RB file from the library. It assigns the correct extension and filename from the begin string in the LF file. In the insert and replace functions, I and R, the local switch /S has to be provided if the module is in ASM, otherwise AL is assumed. The /B switch in the I mode inserts the following module before the first named one; the default is 'after'. In contrast to the LFE program, ALGLIB takes only one module at a time: it is not possible to insert two procedures simultaneously.

Although the program is not very long, about 2 pages and half a page for documentation, a listing and precise description of ALGLIB will be omitted here, but several features should be mentioned. For protection of the library files, ALGLIB leaves them with the write protect and permanent bits set (these are temporarily removed during the ALGLIB operations). This is equivalent to the CHATR CLI command. The program gets its

ARGUMENTS WITH THE COMARG ALGOL COMMAND. THE FIRST SUCH ARGUMENT IS THE FUNCTION (X, F, ETC) FROM WHICH IT CALCULATES A LABEL INDEX. THE PROGRAM TRANSFERS TO THE LABEL, CHECKS THE PRESENCE OF THE MODULES ON DISK AND IN THE LT FILE. THE LF OPERATION IS DONE FIRST, BY USING LINEREAD UNTIL THE PROPER DELIMITER IS FOUND, WRITING THE FILE INTO A SCRATCHFILE, TO WHICH IS ADDED ANY NEW TEXT FILE. THE SCRATCHFILE IS LATER RENAMED AFTER DELETION OF THE LF FILE. AFTER THE LF UPDATING, THE LFE COMMAND IS INVOKED TWICE: ONCE FOR THE INSERT OR DELETE, AND THEN FOR GENERATING THE NEW LT FILE. THE LFE COMMANDS ARE WRITTEN TO THE CLI.CM FILE. SWAPPING TO CLI.SV PERFORMS THE OPERATIONS IN THE CM FILE. AFTER RETURN, THE PROTECTION IS SET ON THE LIBRARY, AND ALGLIB RETURNS TO THE CLI.

END; END;

THE COVER ; ALEPH$_2$

So from some beginning like this we first think of asking the question: "Then what, after all, is bigness and smallness ?"
Plato

OF ALL THE COMMENTS RECEIVED ON NUANCE1, AND THEY ARE STILL COMING IN, NONE HAS MENTIONED THE COVER, OR SUGGESTED AN APPROPRIATE COVER SUBJECT FOR NUANCE2. JOE CELKO SENT THE MELIOR FONT "NUANCE" HEADER USED HERE: CERTAINLY ' DIFFERENT FROM MY HAND-DRAWN CHARACTERS ON THE PREVIOUS ISSUE. BY LACK OF CONTRIBUTIONS ON THE ALGOL SIDE OF THE TITLE COVER, THE EDITOR AGAIN HAD TO SPOUT A LITTLE WHIF OF ERUDITION (REALIZING HE WILL RUN OUT SOON), AND LO! THERE WAS ALEPH-TWO. THIS IS THE SYMBOL FOR THE TOTAL NUMBER OF ALGOL PROCEDURES THAT CAN BE WRITTEN.

G. L. F. P. CANTOR ORIGINATED THE ALEPH SYMBOLS (THE FIRST LETTER FROM THE HEBREW ALPHABET) IN HIS STUDIES OF INFINITE SERIES, SPECIFICALLY TO SEPARATE THE VARIOUS ORDERS OF INFINITY. ALEPH-ZERO, THE SIMPLEST INFINITY, IS THE NUMBER OF INTEGERS. IT IS EASILY SHOWN THAT THE NUMBER OF ODD INTEGERS AS WELL AS E.G. THE NUMBER OF PRIMES, ALSO EQUALS ALEPH-ZERO. HOWEVER, THE NUMBER OF POINTS ON A LINE SEGMENT, E.G. FROM 0 TO 1, AND THUS THE TOTAL NUMBER OF REAL NUMBERS, IS ONE INFINITY HIGHER, INDICATED BY ALEPH-ONE. [DO NOT TRY TO PROVE THIS WITH A PROGRAM HAVING THE DECLARATION REAL (15) ALEPH0; THE PRECISION DOES NOT QUITE MAKE IT!]. ALEPH-ONE IS ALSO THE NUMBER OF POINTS IN A PLANE (OR CUBE!) BY A REASONING, SIMILAR TO THE PROOF THAT THE NUMBER OF PRIMES EQUALS ALEPH-ZERO.

AN ALGORITHM CONNECTS POINTS IN A SUITABLE FUNCTION SPACE, IN A DETERMINED ORDER. THEREFORE, THE INFINITY OF ALGORITHMS IS ONE ORDER HIGHER YET, AND IS ALEPH-TWO. THIS IS ALSO, OF COURSE, THE NUMBER OF ALGOL PROCEDURES THAT CAN BE WRITTEN. THIS ALEPH-TWO IS THE HIGHEST ALEPH THAT MAKES PHYSICAL "SENSE", I.E. HIGHER ORDERS CAN BE DEFINED, BUT HAVE NO KNOWN PHYSICAL INTERPRETATION. AN EASILY READABLE TREATMENT OF THE ALEPHS CAN BE FOUND IN SCIENTIFIC AMERICAN, MAR. 1966, P.112; MORE THOROUGH TREATMENT OF COURSE IN ANY RECENT TEXTBOOK ON NUMBER THEORY, AND FOR A RELATED TOPIC ON INIFINITY MACHINES [THE ONE JUST BEYOND THE ECLIPSE], SEE THE ARTICLE BY A. GRUNBAUM, SCIENCE 159, P.396 (1968).

END ; END ;

ALEX, *continued from p. 4*

```
END DEMPROC;
R
ALEX/V DEMO DEMPROC   Call Alex with verify
 9:18:42 FRI  9 APR  1976 : DEMO
ALGOL/L DEMO
ALGOL/L DEMPROC                Verify output
RLDR DEMO.MP/L DEMO DEMPROC  ALGPROC.LB ALGOL.LB
DELETE DEMO.LS DEMPROC.LS DEMO.MP
DEMO
POP
                                       ALGOL Compiler messages
PROGRAM IS RELOCATABLE

PROGRAM IS RELOCATABLE
                               .TITL    DEMPROC
IN DEMO NO ARGS  <----------  Output from program
R
SPEED DEMPROC.AL      No real changes made in DEMPROC
!UEHSS
R
ALEX              Repeat Alex call, args not needed
 9:22:00 FRI  9 APR  1976 : DEMO
ALGOL/L DEMPROC
RLDR DEMO.MP/L DEMO DEMPROC  ALGPROC.LB ALGOL.LB
DELETE DEMPROC.LS DEMO.MP
DEMO
POP

PROGRAM IS RELOCATABLE
                               .TITL    DEMPROC
IN DEMO NO ARGS
R
SPEED DEMO.AL
!SCONSOLE;$IS:=" ARGUMENT ";$$
!SDEMPROC$I(S)$T$$                    Add argument
WRITE(CNSO,"IN DEMO ");DEMPROC(S)(↑);
!UEH$$
R
ALEX
 9:27:11 FRI  9 APR  1976 : DEMO
ALGOL/L DEMO
RLDR DEMO.MP/L DEMO DEMPROC  ALGPROC.LB ALGOL.LB
DELETE DEMO.LS DEMO.MP
DEMO
POP
                         Recompile revised DEMO
PROGRAM IS RELOCATABLE
IN DEMO DEMPROC S=  ARGUMENT   Output changed
R
```

END; END;

LETTERS, *continued from p. 9*

WORKING. WORSE STILL, THERE HAVE BEEN OTHER PROBLEMS WITH FILESIZE, POSITION, AND THE DISK HANDLER SOFTWARE WHICH HAVE BEEN "BUGGERS" AS WELL. IF YOU WOULD LIKE MORE INFORMATION ON THESE AND OTHER ALGOL ATROCITIES, I'LL GLADLY SEND IT TO YOU.

/** COMMENT:
I SUGGEST THAT YOU FILL OUT THE SOFTWARE TROUBLE REPORT FOR DGC WITH THE BUG ANALYSIS, AND THEN WRITE AN EXCERPT OF IT, SUITABLE FOR NUANCE, SO THAT OTHERS CAN BE ALERTED BEFORE THE BUGS ARE FIXED.
; END **/

END ; END ;

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

N1,Q/A6: On TI700 terminals.

A: Several readers replied on the use of slow-motion carriage return terminals that require delay. Incredibly, the regular TTY handler in RDOS operates with lower case terminals in many applications. Apparently, the device characteristic to change lower to upper case (bit 2) is not set, and the TTY does the translation locally. The Users Group Hardware Section has circuits for two possible modifications. Contact H. Pottinger for details.

Q1: You always give references to a Manual. Which one? The INCLUDE command is not available in our version, in which one is it available? Is there any difference between ENTIER and FIX besides the handling of multiple precision variables? [A. Burkart]

A: The manual most quoted is the Algol Users Manual, 93-52-5 as mentioned in Nuance1. Other manuals are identified more fully. I have not heard of any versions in which INCLUDE was not available; try the command, there was no documentation in my early version, but the command worked (but see also the DOC section, this issue). According to the most recent literature, there is no difference between ENTIER and FIX except for the limitation in FIX to handle only default precision values. As you pointed out, neither does a "real" entier(n) which is defined as the largest integer equal to or smaller than the argument.

Q2: Does anyone have information on business applications for DGC Algol? [J. Celko]

Q3: Can a subset be made from the RDOS library to save core space for programs that use only a few types of commands such as integer arithmetic? [K. M. McClelland]

Q4: How does Algol core usage compare with that of Basic and Fortran? How can Algol core usage be estimated? [R. E. Williams]

A: Combining Q3 and Q4, the reduction in size would be of great interest to users of microNovas; there is some information about the size required, and variations, in the manual, p. E9. Nothing is said about making and using other subsets, and it is not very clear how to go about omitting parts not wanted.

Q5: SOS! ALGOL! SOS!

Is the Algol compiler available in a RB version which could be configured to provide I/O from and to cassette tapes? I have an absolute binary SOS version on paper tape with TTY I/O. [T. R. Churchill]

Q6: How can I get started with Algol on my paper tape system? Interestingly enough, my manual contains instructions on how to operate with paper tape. It does not tell me what I feel is the true answer: you can't. You must have some live interaction with a system as a learning process when you are just getting started. There are too many intermediate tapes required, and the manual warns that some errors (conflicts with ASM symbols for example) will not be detected until late in the process. [J. Pierce]

A: Again, a combination of questions on a single theme, of interest to microNova users. The next issue probably will have Q/A on how to get Algol on the floppy disk. Many users say they are programming microcomputers (8080, 6800?) because the language is so eminently suited for bitpushing etc. I understand from DGC that at the moment, there is no support for Algol compilers except under RDOS, but that all systems will handle the Algol generated RB files.
Added at press time: maybe there is hope, see Owing's letter, this issue.

Q7: Please explain I:=X:=S on p.F10 (manual) where S = string, X = real, and I = integer. What does the COMARG do in the example? [T. R. Churchill]

A: Algol allows automatic type conversion such as in the dual assignment above where a string S (e.g. '29.98') is transferred to a real number X = 29.98 and then to an integer I = 29. This facilitates operations such as X-I which returns the fractional part, 0.98, of X. See the manual p.6-3 for the allowed data conversions. For the COMARG procedure, see p.9-26. It allows you to call a program with arguments. Instead of calling the program by typing its name 'CHECK <CR>' and having the program ask the input on the TTY, the program can read 'CHECK 29.98 <CR>' or even 'CHECK/U 29.98 <CR>' where 29.98 is a string argument to the program name, and /U a switch. In the example, the first call COMARG(1,S) returns S = 'CHECK', the second, identical, call resets S to '29.98' which string is used as shown above. You could modify this program by using COMARG(1,S,BA); COMARG(1,S);. Here BA should be declared a boolean array BA[25]. Without switches to the name, all elements of BA are set false. With switch /U as above, element 20 (i.e. U in A=0,... Z=25) is set to true, and with switch /S similarly, BA[18]=true. The program could use these switches to apply correction factors to X for US or Singapore dollars, and print the new number as letter string in Canadian dollars.

The COMARG procedure is used in system calls such as RLDR/A PROG LS/L 3/K. Another example is discussed in ALEX. In 'ALEX/V DEMO DEMPROC' 4 calls must be made to comarg:

```
OPEN(0,"COM.CM");
AA: COMARG(0,TMP,SW); /*sw is bool. arr. for
 global switch*/
BB: COMARG(0,TMP,SW,ZZ); /*main program*/
CC: COMARG(0,TMP,SW,FINI); /*modules*/ GOTO CC;
FINI: /*final processing*/
ZZ: /*end of Alex*/
```

After AA, string TMP will contain 'ALEX', and SW[21]=true. This call is not required to get the string, but for the global switches. After BB, TMP = 'DEMO' the main program name to be worked on, all SW are false, and exit to ZZ is made when Alex is called without an argument. The loop on CC gets the modules into TMP, one by one, with the local switches such as /S for .SR files. When this loops runs out! transfer is made to FINI for final processing.

Q8: I have had no success in using either VALUE statements or double parentheses. I found that substituting a value field for a local field in the calling sequence has no effect, but that putting it in the parameter field in the called location works exactly right. Can anybody straighten me out? [K. M. McClelland]

A: Due to the deviations in calling between Algol60 and DGC Algol, there is confusion. A procedure (see FIB, this issue) needs to specify its parameters, e.g. that N is an integer of default precision. This is done with one or more statements 'INTEGER N' (identical in form to declarations in the beginning of a program or block), which follow the procedure declaration 'INTEGER PROCEDURE FIB(N)', and precede the block that contains the procedure body 'BEGIN ... END FIB;'. Thus the line containing 'INTEGER (15)...' has two statements that are declarations inside the procedure FIB. With the procedure as shown, the value substituted for N may be changed inside the procedure (not done here), but will not be returned into the main program. Without the statement "VALUE N;" any change occurring inside FIB on N will be returned as changed value to the caller. It is thus essential to omit the VALUE statement if N is an output parameter. See also DOC6, this issue; a future issue hopefully will have a full description of procedure calls (any volunteers for writing this?).

END ; END ;

A RECURSION ON RECURSION

A. VAN ROGGEN

IN NUANCE1, I WROTE ABOUT RECURSION AND MADE A REMARK ABOUT POSITIVE AND NEGATIVE FIBONACCI NUMBERS. HERE THE TABLES ARE TURNED: THE PROMISED FIBONACCI PROCEDURE, AND JUST A REMARK ON RECURSION.

FROM THE RECURSION FORMULA FOLLOWS THAT

F(N-2) = F(N) - F(N-1)

THEREFORE, WITH THE STARTING VALUES F(0)=0 AND F(1)=1, WE FIND THAT F(-1)=1, AND THE FOLLOWING TERMS TOWARDS THE NEGATIVE INDICES: -1, 2, -3, 5, -8, ETC. THE MAGNITUDE OF THE FIBONACCI NUMBER DEPENDS ONLY ON THE INDEX, THE SIGN IS NEGATIVE ONLY FOR EVEN NEGATIVE INDICES. THE PROCEDURE FIBONACCI TAKES CARE OF THIS IN THE "IF N<0" CLAUSE WHICH CHANGES THE SIGNS OF N AND F WHERE NECESSARY. WATCH OUT FOR THE N/2*2 INTEGER COMPUTATION. ALGOL60 REQUIRES ENTIER(N/2)*2, BUT NOT THE DGC ALGOL WITH ITS FUNNY INTEGER MATH (SEE NUANCE1, D11). AT THIS POINT IN THE CODING, N>0 IS GUARANTEED, AND THERE IS NO NEED TO WORRY ABOUT THE RESULTS. THE SAME HOLDS FOR THE ENTIER IN THE NEXT LINE, WHICH DOES NOT NEED AN "OVERLAY" (I.E. A NEW DEFINITION INSIDE THE PROGRAM OR IN THE ALGOL LIBRARY, WHICH TAKES PRECEDENCE OVER THE DGC SUPPLIED PROCEDURE) AS IT WOULD FOR THE GENERAL CASE.

THIS IS THE NEGATIVE INDEX PART OF THE PROCEDURE. REMAINS THE FASTEST RUNNING PROCEDURE WITH THE SHORTEST CODING: ONE STATEMENT ONLY! THIS IS INDEED SHORTER CODING THAN THE RECURSIVE FIB1, ESPECIALLY WHEN THE CONSTANTS ARE USED IN THE STATEMENT RATHER THAN DECLARED AS SEPARATE LITERALS. THE LATTER WAS DONE FOR DOCUMENTATION PURPOSES, AND DOES NOT INFLUENCE THE SIZE OR RUNNING TIME OF THE PROCEDURE. THIS TIME IS SHORTER IN THE RECURSIVE FIB1 WHEN THE DEFAULT VALUES (N<2) ARE USED, BUT NOT IN THE GENERAL CASE. FOR LARGE N, FIB1 BECOMES VERY SLOW WHILE FIBONACCI HAS A CONSTANT RUNNING TIME. A SIMILAR CONDITION HOLDS FOR THE NON RECURSIVE FIB2; THE CROSSOVER TIME DEPENDS ON THE HARDWARE USED.

THERE ARE NOW THREE CANDIDATES: FIB1, FIB2, AND FIBONACCI, AND AGAIN THE QUESTION WHICH ONE IS BEST. THERE IS NO SUBSTITUTE FOR KNOWLEDGE. IF YOU KNOW THE PROBLEM TO BE SOLVED, SUCH AS HERE SOME BACKGROUND ON INFINITE SERIES, CONVERGENCE LIMITS, AND THE LIKE, THE FIBONACCI TYPE SOLUTION CAN BE WRITTEN, AND SHOULD BE INCORPORATED IN THE LIBRARY FILE. HOWEVER, LACKING THIS KNOWLEDGE OR HAVING NO TIME TO STUDY THE PROBLEM DEEPER, THE SECOND BEST IS THE RECURSIVE FIB1. NOT MERELY BECAUSE IT IS ELEGANT, BUT BECAUSE IT IS A DIRECT TRANSLATION OF THE DEFINING EQUATION AND FOR CASES JUST SLIGHTLY MORE COMPLEX THAN FIB1, THE TIME FOR TESTING AND DEBUGGING MAY EXCEED THE RUNNING TIME. WHEN THE PROCEDURE HAS TO RUN FAST, IN THE LONG RUN IT IS BETTER TO STUDY THE PROBLEM AND TO ARRIVE AT FIBONACCI (OR WHATEVER NAME THE PROBLEM HAS), RATHER THAN TO WRITE A VERSION OF THE NONRECURSIVE SOLUTION, WHICH IS NOT VERY FAST EITHER.

FROM GERMANY CAME THE ONLY COMMENT ON RECURSION, BUT NO SUGGESTED PROCEDURE. A. BURKART WROTE:

"AN IMPORTANT DIFFERENCE BETWEEN ALGOL AND FORTRAN IS THE AVAILABILITY OF RECURSIVE PROCEDURES. BUT ON THE NOVA YOU SHOULD NOT USE RECURSION! WE HAVE TESTED THE FIBONACCI ALGORITHM TO FIND THE DIFFERENCES IN COMPILATION AND RUNNING TIME BETWEEN ALGOL AND FORTRAN, AND BETWEEN THE RECURSIVE PROCEDURE AND ONE USING A LOOP. THE RECURSIVE PROGRAM NEEDS 250 SECONDS FOR N=25! HORRIBLE. THE TABLE BELOW GIVES IN SECONDS THE RUNNING AND COMPILATION TIME FOR THE FORTRAN AND ALGOL PROCEDURES (USING INTEGER OR REAL ARGUMENTS OF VARYING PRECISION). RUN TIME 0 MEANS THAT THE RESULT APPEARED IMMEDIATELY ON THE DISPLAY. THERE IS AN IMPORTANT DIFFERENCE IN COMPILING AND RUNNING TIME BETWEEN ALGOL AND FORTRAN. BUT THE ADVANTAGE IN USING ALGOL IS THAT YOU CAN USE MULTIPLE PRECISION UP TO 15 WORDS, WHEREAS IN FORTRAN YOU CAN USE ONLY SINGLE PRECISION. THE LARGEST POSSIBLE NUMBER FOR PRECISION 15 IS ALSO GIVEN BELOW. TO COMPUTE THIS NUMBER WITH THE RECURSIVE PROCEDURE, I THINK ONE COULD WAIT AT LEAST A WHOLE LIFETIME."

```
FIB(324) = 230414835855241682622209064894201807 5
           1016174667804967905736902899 68
```

SO FAR, BURKART'S COMMENTS ON RECURSION (SEE ALSO Q/A AND DOC). HIS RESULTS ARE VERY INTERESTING: THEY SHOW THAT ALGOL IS SLOWER THAN FORTRAN IN BOTH COMPILATION AND IN RUNNING. I HAVE NOT SEEN HIS SOURCE TEXTS, BUT ASSUME THAT THEY ARE "EQUAL". THE COMPILATION DIFFERENCE IS NOT SERIOUS FOR MOST APPLICATIONS, BUT I AM SURPRISED THAT THE REAL (2) ALGOL IS SLOWER THAN THE FORTRAN. I HAVE ONLY COMPARISONS ON TWO MACHINES OTHER THAN THE DGC, AND IN BOTH CASES, ALGOL WAS CONSISTENTLY FASTER FOR GENERAL PROGRAMS. THE ADDITION TIME SHOULD BE IDENTICAL FOR THE TWO LANGUAGES AND MOST LIKELY, THE COMPUTER SPENDS MOST OF ITS TIME ON THOSE STEPS IN THE FIB2 PROCEDURE. IS THERE AN EXPLANATION FOR THIS? THERE MAY BE A HARDWARE DIFFERENCE; ON AN 840/32K/HMPYD, COMPILATION TAKES ONLY 26 TO 27

RUNTIME OF FIBONACCI PROCEDURES (sec)

	FORT	A L G O L				
N	IV loop	I(10) loop	I(15) loop	R(2) loop	I(1) rec	R(2) rec
10	0	0	2	0	0	3
16	0	1	-	0	2	45
17	0	1	-	0	4	72
18	0	1	-	0	8	120
19	0	1	-	0	14	-
20	0	1	-	0	22	-
25	0	1	5	0	250	-
100	0	7	15	1	-	-
200	0	11	26	2	-	-
300	0	-	32	3	-	-
324	0	-	33	3.5	-	-
COMP TIME	25	37	35	36	38	38

```
/***FIBONACCI***/
INTEGER PROCEDURE FIBONACCI(N);
VALUE N;INTEGER N;
COMMENT RETURNS FIBONACCI NUMBER F(N)=F(N-1)+F(N-2),
PHI=GOLDEN RATIO, W5=SQRT(5);
BEGIN
LITERAL PHI(1.618034),W5(2.236068);INTEGER F;
F:=1;IF N<0 THEN BEGIN N:=-N;IF N/2*2=N THEN F:=-1 END;
FIBONACCI:= F*ENTIER(PHI↑N/W5+0.5);
END FIBONACCI;
```

seconds for small Algol programs like these (comp. 37 in the Table) while the running time also is shorter. For the I(15) loop case, with N= 20,25,100 I find 3,3, and 9 sec. With I(1) recursive, N= 20,25 comes out at 7 and 147 sec, and I(15) rec. increases these times to a phenomenal 813 and 9015 seconds. There is thus a severe penalty in running time for extended precision. By the way, the running time for FIBONACCI is 0, with or without extended precision. The calling program and the procedure, or at least one version thereof, are shown for comparison.

I have received no other Fibonacci procedures, but the challenge is still open! In his comments on Nuance1, M. J. Mardesich remarked that the statement on language comparisons could be interpreted to imply that SNOBOL is non-recursive, while the power of this string handling language actually derives to a large part from its recursion. Correct, of course! I intended to say that SNOBOL lacked some of the advanced features from Algol such as no real numbers, a restricted logical branching, etc. Is there interest in a more detailed description of various languages and the differences with Algol or in procedures where recursion cannot be avoided? Let me know, or better yet write a contribution for Nuance.

```
TYPE TRYFIB.AL
BEGIN
/*TRY VARIOUS FIBONACCI PROCEDURES*/
LITERAL CR("<15>");
EXTERNAL STRING PROCEDURE NMTIME;
EXTERNAL INTEGER (15) PROCEDURE FIB;
INTEGER (15) R;INTEGER P;
OPEN(2,"$TTO");
FOR P:=20,25,100 DO
 BEGIN
 WRITE(2,(NMTIME),CR);
 R:=FIB(P);WRITE(2,R,CR,(NMTIME),CR);
 END;
END
R

TYPE FIB.AL
INTEGER (15) PROCEDURE FIB(N);
VALUE N;INTEGER N;
BEGIN
INTEGER (15) F,F1,F2;INTEGER I;
F:=F1:=0P15;F2:=1P15;
FOR I:=1 STEP 1 UNTIL N DO
 BEGIN
 F:=F1+F2;F2:=F1;F1:=F
 END;
FIB:=F
END FIB;
R
```

Note multiple precision literals F1 and F2, and the () around procedure NMTIME when it is used without arguments.

```
 8:50:32
6765
 8:50:35
 8:50:35
75025
 8:50:38
 8:50:38
354224848179261915075
 8:50:47
```

END; END;

Reviews:

J. J. Donovan: "Systems Programming", McGrawHill 1972

From Project MAC at M.I.T. comes this book on the ins and outs of systems programming. According to the Preface, the book is meant not only for students, but also serves as a software course emphasizing practical issues. This it does admirably, starting each Chapter with a short summary of items to be treated, in addition to a complete general background of systems programming in the first Chapter. Anyone who has worked with a computer and gone through the various steps of editing, compiling, assembling, etc. will be able to follow in detail the intricacies of these processes, often so nonchalantly typed on the console as e.g. ALGOL TEST; RLDR/M TEST PROC ALGLIB; TEST, or with even less thought as ALEX (see elsewhere in this issue of Nuance).

Although the book does not specifically use Algol, it describes many Algol features, and of special interest to Nuance readers will be the general description of the features of PL/I. Many of the DGC extensions to Algol follow the PL/I conventions: for example the pointers, based variables, and accociated procedures such as ALLOCATE. These are explained quite thoroughly in the book, and thus could help those Algolisti who care not for compilers and loaders, but wish to bring more nuances into their Algol programs.

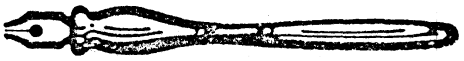
DGC: "RTIOS Users Manual", 93-95-2

Remember the plight of the library clerk who had led a sinful life, landed in the Library of Hell, and found the message that the only way to redemption was to translate a Chinese classic into English? Not knowing any language but English, he went to the dictionary section where were found two volumes: English to Sanskrit, and Sanskrit to Russian. The puzzle this fellow had was minor compared to the translation needed for the RTIOS manual, which consists of an incredible number of undefined terms, cloaked in grammatical vagueness, and decorated with errors rather than examples.

Chapter 1 is the general introduction; the lack of specifics on possible variations makes the chapter too short to be useful. Chapter 2 describes the Fortran calls. Perhaps this Chapter will be clear to Fortran users, many of whom are accustomed to imprecise definitions, but the formal parameters (in Fortranese: dummy variables) are described in the same dummy terms, and much experimentation is required with the system to know what it is doing (assuming that both software and hardware are compatible with the system).

Chapter 3 has the ASM calls to the RTIOS system; this is the Chapter required to make the hardware Algol addressable. The fun starts with the interpretation of pointers (Algol), pointers (ASM), address, etc., and then widens out to guessing the various uses of channels (Algol channel, RDOS channel, RTIOS logical channel, hardware channel number) and culminates in the mystery of devices (device code, device number, subdevice number). None of these terms is defined properly or related to the specific pieces of hardware. On an A/D multiplexer is an analog channel the same as a subdevice number? Due to the large variety of "gadgets" that can come with the computer, it is perhaps understandable that the same nomenclature cannot be used universally: an analog input line and a multiprocessor link are quite different. But whatever terms are chosen, they should be identified, and any differences made clear.

Chapter 4 contains the sysgen procedure; in contrast with some other sysgens (e.g. RDOS Appendix E), this one is quite clear. There is a program error in the order of the questions (related to scope control), but this is a minor point. Worse is that you must have a Fortran system to do an ASM sysgen: the Fortran source code is provided, not the RB files. By the way, a clever Systems Engineer recently has made a nice, improved version of this sysgen program, which eliminates the use of the extra loading tapes. Ask G. M. Sipe (DGC, Blue Bell PA) for information.



THE PULITZER PRIZE

DGC: "HOW TO LOAD YOUR NOVA RDOS SYSTEM", Manual 93-188-0.

After almost ten years of publishing manuals, DGC has finally done it: a "How to.." book! And in stark contrast with most "How to do it better with less pain" books (and with some manuals that contain the pains only), this sysgen manual is almost perfect. It is clear, well written, has ample cross references, summary tables, and -last but not least- when all the steps are followed one by one, the system really works! Congratulations to the anonymous author who transformed the old Appendix E of the RDOS manual to a useful Work of Distinction. This manual relates to the sysgen of RDOS Rev 5, of which the whole procedure has been improved and updated. No more mysteries about which paper tapes to LOAD and which ones to XFER: now all are in the LOAD/DUMP format, indicated by the tape name extension .DP (not all new tape names are listed correctly in the manual), while the few essential absolute tapes (.AB) have the correct loading (MKSAVE) instructions clearly spelled out. Only one part is not quite clear: some tapes are marked with part 1, 2, etc, and must be loaded in that order, while other tapes are similarly marked but are not segmented and can be randomly loaded. The console message is always correct, the text (or tape name) is not. In the chapters on cassettes and magnetic tape chapters it would be helpful, at least in mixed media systems, to have a listing of the BACKUP file commands. Which ones have to XFER and which ones to DUMP for proper startup? Are switches needed? But these are just minor faults and hardly detract from the delights of the improved system and manual.

One final suggestion: Why not do away with the Algol appendices and replace them with new books of this quality???

Alle Rechte, insbesondere das der Übersetzung
in fremde Sprachen, vorbehalten....
Springer-Verlag

R. Zurmuhl: "MATRIZEN", 4. Auflage, Springer Verlag, Berlin 1964.

If you don't recognize the language of the above quote, don't read this review. To my knowledge, no translation of this textbook on matrix theory is available. Pity, because the book is more Urgründlich than most. Like almost all European books, it has examples to elucidate the text, but no problems for the reader to work on. Sometimes I wonder about the cause for this typical difference. The reason for this review? In contrast to the earlier editions of this book, this one uses Algol programs (especially on eigenvalues and eigenvectors) and gives a number of clear descriptions of algorithms for calculating e.g. inverse matrices. The book also has changed from the old "German" letters to bold type representation of matrices, so that, especially for the programs, only a smattering of German is needed for comprehension. In view of the above quote, I hardly dare to give an example from the book, but if there is interest in this subject, Nuance will carry some matrix operation procedures and show how the use of pointers can improve the running time beyond belief.

PAGE FILL PROCEDURES

Anyone, working on a newsletter, abhors a vacuum - at least on paper. Therefore, some procedures to fill the last page of this issue.

MATRIX INVERT

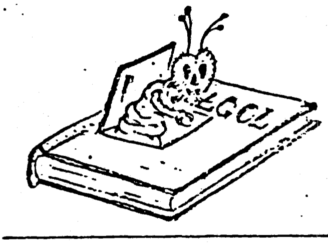
The next issue will have a description of the use of pointers and based variables. The "working example" will be a procedure that inverts a matrix by the pivot method. So as not to get confused about the pivotal operations and mix them up with the pointer tricks, the "straight" Algol code is shown here. The outer block tests the matrix for squareness, checks that the lowest index is "1", and finds the size. The inner loop does the pivoting and permutation of the rows and columns. This is by no means the best procedure for inversion (What happens when the matrix is singular?) but will serve well to show pointer operations.

CLI OPERATION FROM ALGOL

The call CLI("LIST/S -.AL") from an Algol program that declares CLI as an external procedure, will execute the corresponding CLI call as if the command were typed on the (background) console. EXEC.SR must be made as shown in Nuance3.

```
/***CLI***/
PROCEDURE CLI(CMD,FRG);
VALUE CMD;STRING (100) CMD;BOOLEAN FRG;
/* EXECUTES CLI COMMANDSTRING FROM ALGOL */
BEGIN
LITERAL FNM("FCLI.CM"),NM("CLI.CM"),SP("<15>POP<15>");
EXTERNAL PROCEDURE EXEC;STRING S;
IF FRG THEN S:=FNM ELSE S:=NM;
IF CMD<>"" THEN
 BEGIN
 OPEN(7,S);WRITE(7,CMD,SP);
 CLOSE(7);EXEC("CLI.SV");
 END;
END CLI;
```

```
PROCEDURE OINVERT(A);
REAL ARRAY A;
COMMENT INVERTS MATRIX A IN PLACE WITH PIVOT METHOD
WRITTEN IN PURE ALGOL, NEWER INVERT USES POINTERS;
-
BEGIN
INTEGER N;EXTERNAL INTEGER PROCEDURE.DIM;
IF NOT (DIM(A)=2 AND LBOUND(A,1)=1 AND LBOUND(A,2)=1)
 THEN BEGIN WRITE(2,"DIM. ERROR");GOTO A1 END;
N:=HBOUND(A,1);IF N<>HBOUND(A,2) THEN
 BEGIN WRITE(2,"MATRIX NOT SQUARE");GOTO A1 END;
BEGIN
INTEGER ARRAY Z[1:N];REAL ARRAY B,C[1:N];
INTEGER I,J,K,L,P;REAL Y,W;
FOR J:=1 STEP 1 UNTIL N DO Z[J]:=J;
FOR I:=1 STEP 1 UNTIL N DO
 BEGIN
 K:=I;Y:=A[I,I];L:=I-1;P:=I+1;
 FOR J:=P STEP 1 UNTIL N DO
  BEGIN
  W:=A[I,J];IF ABS(W)>ABS(Y) THEN  BEGIN K:=J;Y:=W END;
  END;
 FOR J:=1 STEP 1 UNTIL N DO
  BEGIN
  C[J]:=A[J,K];A[J,K]:=A[J,I];A[J,I]:=-C[J]/Y;
  B[J]:=A[I,J]:=A[I,J]/Y
  END;
 A[I,I]:=1/Y;J:=Z[I];Z[I]:=Z[K];Z[K]:=J;
 FOR K:=1 STEP 1 UNTIL L,P STEP 1 UNTIL N DO
 FOR J:=1 STEP 1 UNTIL L,P STEP 1 UNTIL N DO
  A[K,J]:=A[X,J]-B[J]*C[X];
 END;
L:=Ø;
BACK: L:=L+1;
AGAIN: K:=Z[L];IF K=L THEN GOTO TEST;
PERMUTE: FOR I:=1 STEP 1 UNTIL N DO
 BEGIN W:=A[L,I];A[L,I]:=A[X,I];A[K,I]:=W END;
Z[L]:=Z[K];Z[K]:=K;GOTO AGAIN;
TEST: IF L<>N THEN GOTO BACK;
END;
A1: END OINVERT;
```



DOC BUGS CRAWLS OUT :

NOTES ON ALGOL DOCUMENTATION AND BUG REPORTS

D0 - NUANCE1

Nuance is produced with a one-pass typesetter, proofreader, printer and producer. This shows in the errors:
*p.6, in D9, "   . E " should be "###.##E##", and similarly, change "   R8" to "###R8".
*p.10, in ASM procedures, change "TME(H<M<S)" to "TME(H,M,S)".
*p.6 The appearance of apparantly in D3 is an apparent appendant error. Vive l'Anglais. There are too many such typos.

D1 - ALGOL MANUAL 93-52-5

*p.9-36 ff. The word "filenumber" in the CMM section should be changed to "channel", and e.g. "File 0" (p.9-43) "Channel 0".
*p.9-45 has lines reversed in the first example block: "Literal MINRES" must be declared before the arrays A,B. These arrays are "integer", not "literal".
*p.6-10 The definition of "uc" states "cannot contain another IF clause". This should be "cannot be", because with BEGIN ... END any legal IF clause can be nested. See N1,D12. Also change "contain" to "be" on line 5, p.6-11.
*p.7-19, line 3. Declare "based array i[-1:5]"
*p.7-20, example. Change to LOOP: IF p->i=0 then goto EXIT1 else if (p:=p+1)->i= key then goto EXIT2 else goto LOOP.
*p.C15 add "desc1" and "desc2" in the coding of CVST
*p.E1, bottom. Substitute "not j:= i+2.0" in comment.

D2 - LABEL PARAMETERS

When using label parameters in a function (typed) procedure, and when an exit is made to this label, the value of the procedure is not returned. For example: INTEGER PROCEDURE TEST(A,LB); INTEGER A; LABEL LB; BEGIN TEST := IF A<0 THEN 0 ELSE A2; IF A<0 THEN GOTO LB; END TEST; and the call ... X:= TEST(-5,ZZ); ... ZZ: WRITE(CH,X); is made, no "0" is written at ZZ. It is more reliable to use a boolean error parameter, such as INTEGER PROCEDURE TEST2(A,ER); INTEGER A; BOOLEAN ER; TEST2 := IF (ER:=A)<0 THEN 0 ELSE A2; which also gives shorter coding.

D3 - RDOS date, time

Some Algol programs have to read the file information (UFD in RDOS terminology). There are entrees for dates (1 word each), and time (1 word for hour and minute). The recipe is TIME := 256 * H + M; DATE := JD - 2439857; where JD is the Julian date number for the specified day, with an offset equal to the Julian date number for Sunday, 31 Dec 1967. If a routine is required to transform from the JD value to the Gregorian calender date, or vice-versa, I recommend the Algol procedures JULDAY and GREGDAY. These were submitted to the Users Group Library in the middle of 1975, and they will be listed any year now, certainly before 1985.

D4 - SUBSCRIPTED LABELS

The call READ(CH,X,LBL[-2]); does not work when end of file transfer has to be made to label LBL[-2]. An unsubscripted label is OK.

D5 - BYTEWRITE to devices

Linefeeds are treated differently from other bytes on some devices. If SP= <40>, CR = <15>, and LF = <12>, the three strings S1 := "CR, LF, LF, ..."; S2 := "CR, CR, CR, ..."; S3 := "SP, SP, SP, ..."; give the following results: BYTEWRITE(TTO, PS->x, 5) and BYTEWRITE(CAS, PS->x, 5) generate the proper bytes, but when the cassette file is typed later, (e.g. TYPE CT0:0), no LF's are typed. The PS in the example is the address of any of the strings.

D6 - PARAMETER MATCHING [R. Gavare]

The manual states (p.8-7) that "the number of actual parameters must match (equal? Ed.) the number of formal parameters", and on p. C-4 where n and m are these numbers, "if n=m all parameters will be passed, else if n<>m the lesser number will be passed, else if n=0 or m=0 none will be passed". It is not expressed quite so algolistically! Fortunately, the latter statement is correct. The actual parameters not passed into the procedure are initialized to zero, false, or the null string (depending on their type) in the procedure. This allows such niceties as STRING PROCEDURE NMDATE(DAY, MON, YR); which can be called as NMDATE(5, 1, 1976); to return "MON 5 JAN 1976" or simply as NMDATE; without any parameters, to return the string corresponding with the day the call is made. Another example is shown in ALEX, this issue. The string procedure DEMPROC sees S = NUL if there are no parameters, and uses this as test. Note that, if the main program also has defined NUL, the call DEMPROC(NUL) still would print "no args".

D7 - OUTPUT [R. Gavare, D. Down, R. Burkart]

See also N1, D9. Caution is required for real numbers. At least two "##" are needed to print an exponent (#.##E is interpreted as 1.23E where the E is just a character rather than the exponent) and ".##E##" leads to disaster (trap in a mapped system). If the last argument is an error label, the field counter goes haywire. The format string must have at least one "#", OUTPUT(1,"<15>",A) gives an endless loop.

D8 - ALLOCATE [H. W. Eber]

See also N1, D2. Allocate does not re-use previously freed space. ALLOCATE(P1,100); ... FREE(P1); ALLOCATE(P2,98); assigns new space to P2 and does not "overlay" the area first used by P1.

D9 - WRITE integer, pointer

POINTER P; INTEGER I; I := P; WRITE(1, I, "=", P); which gives the result e.g. '17308=41634' in decimal and octal. The secret? Pointers can think only in octal and print accordingly. This is handier than the OUTPUT (see N1. D9).

D10 - BYTE TRANSFER [H. W. Eber]

BASED INTEGER BI; BASED STRING BS; STRING S; POINTER P; LITERAL L("ABC"); P := ADDRESS(S); A: SUBSTR(P->BI, 3, 5) := L; B: SUBSTR(P->BS, 3, 5) := L; The two statements A and B result in differing byte contents of S. A gives 'xxABCx...' while B gives 'bbABCx...' where x is an arbitrary byte, and b a blank space. Spaces are substituted for the earlier bytes in B, although these bytes are not specified to be filled. Then try C: WRITE(1, S); D: WRITE(1, P->BS); E: WRITE(1, P->BI);. Statement E always works and prints an integer. C prints the string only after SETCURRENT(2,6); and D prints the string before the SETCURRENT command, but not afterwards!

D11 - SYSTEM ERROR CODES

ASM procedures with system calls may use the error returns. If they do, an error code is returned in AC2. If two or more simultaneous error conditions exist, only one code is returned. There is no documentation on the priority of the error tests: this has to be tried the hard way, and procedures should be written with this uncertainty in mind if the value of one or more of the codes is important. A similar uncertainty happens at a "higher" level, and is partly the cause of such mysterious errors such as "File does not exist: MYPROG.SV" when MYPROG.SV is the actual program that generates the message!

D12 - BOOLEAN VALUE [D. Down]

All system calls as well as the Algol COMARG return 0 for false and -1 for true. However, B := TRUE; results in +1 for B, and when this B is used as the actual parameter for an ASM procedure, it acts as FALSE. Presumably, because of this same bug, boolean values other than 0 or 1 are not handled properly: IF BV THEN ... gives true for any nonzero BV. But IF NOT BV THEN ... is false only if BV = 1. Any other nonzero value has the same effect as zero; this includes -1 which is the gospel "true" in all other systems. Curiously, the compiler code generates a zero or one literal for B := false or true, rather than an integer value. The resulting extra shuffles with .LP are not very efficient.

D13 - TRACE [K. M. McClelland]

TRACE.SV does not work properly when the allocated storage is large enough to reach S+167, i.e. with actual stack pointer offset >0. The listing just repeats the storage at S+0 over again.

D14 - OVERLAY [J. Isaak]

In older Algol versions, the wrong message'illegal overlay number' can be fixed with OVOPN+5 / 21001 23001

D15 - RESIDUE < EPS

In computational loops, an exit is often made after a comparison of a calculated residue with a small literal or value such as EPS := 1E-6; Although real numbers with precision (2) can be represented from 1E-78 to 1E75 (Manual, p.4-5), this does not hold for a comparison of small numbers which is limited by the internal representation. Real (2) uses 24 bits for the mantissa, and comparisons should not be made below the least bit (5E-8 or 1E-7 to be safe). With REAL (3) the mantissa has 40 bits, or 1E-13. A quick rule of thumb is to take the number of hexadecimal digits in the mantissa as the negative power of ten to which comparisons can be made [i.e. 1E-6, 1E-10 for REAL (2) and (3), respectively]. This estimate is conservative: for REAL (15) there are 58 hex digits or 232 bits, which is about 1E-70.

D16 - BYTE/LINE READ/WRITE

The specification of the bytecount parameter is INTEGER VALUE in BYTEWRITE and BYTEREAD, but it is INTEGER in the equivalent LINEREAD and LINEWRITE. In case of any error in the LINE procedures, a changed value will be returned, this should thus not be used with literals as the actual parameters. The manual describes this correctly, but does not emphasize the vital difference.

D17 - INCLUDE

Do not use other Algol code on the same line after INCLUDE. The compiler will not remember such text. See N1, D10 which perhaps is the same type of error.

D18 - GLOBAL LABEL [R. Gavare]

A jump to a global label does not work: BEGIN INTEGER I; BEGIN INTEGER J; GOTO L; END; L: END; gives erroneous complaints from the compiler. Inserting the declaration LABEL L does not help.

D19 - FORWARD DECLARATIONS [R. Gavare]

Each identifier must be declared before it is being used, in DGC Algol forward declarations are not allowed. This gives trouble when two or more procedures must call each other. A fix can be made often by using external procedures.

D20 - GLOBAL ARRAYS [A. Ambler, R. Gavare, M. Berthoud]

Procedure references to global integer array elements are faulty (RDOS2.0). This can be circumvented by using based arrays and a pointer. The assignment I := SIZE(A); referring to global INTEGER ARRAY A[10] is flagged as dimension error.

D21 - Mystery ERRORS [A. Burkart]

If an incomprehensible error message occurs, look for missing double quotes in all lines previous to the flagged error line.

D22 - INTEGER ARITHMETIC [A. Burkart]

REAL X,Y,Z,P; X:=Y:=Z:=3.5; X := X - ENTIER(X); Y := Y - FLOAT(ENTIER(Y)); P := ENTIER(Z); Z := Z - FLOAT(P); Burkart writes that, in contrast to all Algol conventions and logical expectation, the result of the above is X = Y = 0.0, and only Z = 0.5 Specifically, FLOAT(ENTIER(Y)) should be 3.0.

It doth amount to one more than two
Which the base vulgar do call three.
Shakespeare

Be careful with INTEGER operations!!
A. Burkart

[Perhaps in Burkart's honor I should quote Goethe rather than a Britisher. Iff my memory serves me, there is a similar numbers remark somewhere in the Farbenlehre, but I don't have a copy nor the time to look it up properly]. The only way I found around such problems (and it cures this one too) is to make an "overlay" procedure, which defines ENTIER properly. This is easily done (below). When its RB is located in the Algol library, any program that declares ENTIER as external integer procedure will use the homemade version rather than the wrong built in one. Looking at the coding, DGC's entier is not even a procedure, but just a shuffle with FMOV 1,1 which - if this is equivalent to the integer MOV instruction - only wastes time. When the external declaration is made, a proper @CALL to ENTIER is made.

```
/***ENTIER***/
INTEGER PROCEDURE ENTIER(X);
VALUE X;INTEGER X;
COMMENT: builtin procedure is wrong (=fix);
BEGIN INTEGER H; H:=FIX(X);
ENTIER:=
H - IF SIGN(X)=-1 AND H<>X THEN 1 ELSE 0;
END ENTIER;
```

D23 - ARRAY DECLARATIONS [M. Berthoud]

The wrong declaration INTEGER ARRAY [1:5] A; is not flagged as an error.

END of ;