THE SPECIAL CHARACTER SET

APL SIG DECUS

APRIL 1985 No. 9

FLYING THE JOLLY ROGER

Douglas Bohrer
Bohrer and Company
Wilmette, IL 60091

Your view of piracy depends on where you are when they attack. Historical views on piracy changed just as modern views on software piracy change depending on whether you are a target or not.

In Queen Elizabeth's time, English pirates were viewed as patriots who had begun the war with Spain a little ahead of schedule. Some of the loot from this piracy paid for equipping the fleet that fought the Spanish Armada in 1588. The Spanish started the problem by capturing a peaceful English trading fleet in the Caribbean. They enslaved the captured sailors for daring to break the Spanish Crown's trading monopoly, enfuriating sailors back in England.

The Spanish viewed these pirates as bandits who were murdering their people and stealing their money. They forgot all about looting the English trading fleet. With a big investment in America, the Spanish were livid when their profits were stolen before the money reached Spain. The English Crown's tolerant attitude towards pirates made the Spanish even madder.

At first piracy was not a problem to the English. English pirates only attacked Spanish, and later French, shipping. Soon the French thought the idea was worth imitating so they started piracy against the English and Spanish. Gradually the English built up their own colonies in America. More and more English ships sailed between Europe and America. These ships were carrying valuable cargo and the pirates noticed they were not as heavily guarded as most Spanish ships. So pirates began attacking everybody's ships regardless of nationality. Ultimately the English navy had to make a large effort to suppress piracy and stop the attacks.

Today the nature of piracy is technical and nonviolent. Nobody gets killed when you copy a diskette. The owner of the software losses nothing physically. He still has the original. From the view of the user, piracy harms nobody and helps the pirate. Besides, the software vendors started it all by setting their price too high relative to the cost of floppies. Piracy is OK if you're not being attacked.

Software vendors, on the other hand, feel like they're being robbed. They spend a lot of money developing products. They recover this cost by setting the price of the software far higher than the cost of the diskette or tape it's written on. Pirates are costing them sales. When their ships didn't come in, the Spanish could make a good guess on what happened and how much they lost. Software vendors have no good way of knowing how much they're losing. Most think it's a lot.

Today's software pirates should remember the experience of the English and Spanish. Many people in the computer field, including pirates, dream of starting their own business. Many companies in other fields consider selling software as a sideline. Their view of piracy will change once they're attacked. Will you have to wait until pirates attack you to change your mind?

STRONG DEC COMMITMENT TO APL

Douglas Bohrer
Editor, The Special Character Set

Dave Quigley, Digital's APL Product Manager, emphasized the company's strong commitment to APL in a birds-of-a-feather session he requested in Anaheim. About 20 people crammed into the APL suite to discuss "DEC's Commitment to APL". Dave's opening remarks and the following discussion lasted an hour.

Dave expressed in the strongest possible terms short of a product announcement DEC's commitment to work on signifcant enhancements to VAX-11 APL. Specifics about the features to be added have been discussed with several potential customers who signed nondisclosure agreements. I asked if that meant that these enhancements had been discussed over pizza and he responded "VERY DEFINITELY". The timing for this new version release is probably fall of 1985. Dave expects to be making a "significant presentation" to DECUS at the Fall '85 Symposium at Disneyland.

Dave also said that DEC is working on an APL character set for the VT200 series video terminal. The timing for this product is possibly late summer or early fall of 1985.

Dave was so expansive that he was even willing to talk, very briefly, about APL-11. He said that he would like to get some mail indicating the level of interest in APL on the 11's and PROs.

In additon to being APL Product Manager, Dave Quigley is also the APL SIG DEC Counterpart. His phone and address are on page 2.

CONTENTS

THE SPECIAL CHARACTER SET
April 15, 1985 No.9

The Special Character Set is the official publication of the APL Special Interest Group of the Digital Equipment Computer Users Society. Unsolicited manuscripts are desperately sought by the editor.

For subscription information and an application, contact:

Membership Services
DECUS
249 Northboro Road BP02
Marlboro, MA 01752

STYLE SHEET

by Douglas Bohrer
Bohrer and Co.
903 Ridge Rd Suite 3
Wilmette, IL 60091

We look forward to your contribution to The Special Character Set. Your cooperation on a few matters of style would help get the newsletter out on time. The newsletter staff is all volunteer and the time you save us is appreciated.

Set-Up

Basically, your contibution should look like this article. Use black ink. Type your text single spaced. Use only one side of each sheet of paper. Skip one line between paragraphs. Set your margin at 4 3/8 inches wide. For type with 10 characters per inch, you would have 43 characters per line. For type with 12 characters per inch, you would have 52 characters.

Titles

Please start the title of your article on the left margin using all capital letters. Skip one line, then put your name, title, firm and mailing address. (You may omit your title, firm and mailing address for reasons of modesty, privacy, shame or whatever.) Sub-heads in your article should begin at the left margin with a blank line above and below them.

Letterhead will not be reproduced. Drawings should be either 4 3/8 or 9 inches wide.

Commercial Guidelines

The newsletter will follow strictly the guidelines related to the non-commercial nature of DECUS. If you have any questions on these, please contact the newsletter editor or the DECUS office.

SUITE TERMINAL

The terminal for our hospitality suite in New Orleans will be an LA12-AB with APL keycaps provided by Suzanne (Rusty) Skinner (617-493-7683), DEC's new marketing guru for the Correspondent. The LA12 comes standard with an APL character set. APL keycaps are slightly extra.

APL IN NEW ORLEANS

Susan M. Abercrombie, Symposium Coordinator

The APL SIG has scheduled four sessions for the Spring 85 DECUS Symposium, and will offer demonstrations of APL in our SIG suite.

Doug Bohrer (Bohrer and Company) will answer the age-old question 'Why Use APL?' This is an introduction to the language, and will give the novice some insight into the power and versatility of APL, and some of the applications for which it is especially well suited. Don't be frightened by special characters--learn to be one.

There will be a panel discussion on the various APL implementations in the public domain, available from the DECUS library for a nominal charge. If you have had trouble trying to use one of these interpreters, or if you are curious about them, bring your questions here.

Bob Awde (General Mills) has rebuilt APL-11 to utilize I/D space on RSX-11M-PLUS. He will describe the benefits and the technical problems of this effort. In addition, he plans to present information on how to generate the APL character set on VT200 series terminals.

Dave Blickstein (DEC) will present a DIGITAL-sponsored session on VAX-11 APL. He will focus on the status of VAX APL and plans for the product, with particular emphasis on its unique features, as well as a discussion of internals.

The APL SIG will have a suite equipped with an APL terminal. Demos of APL will be provided there. Look for a schedule of times in the new UPDATE.DAILY publication, or check the list on the suite door.

See you in New Orleans!

NEW APL SIG STANDARDS REP

Douglas Bohrer
Editor, The Special Character Set

The APL SIG has signed a new steering committee member, Dan Esbensen, as our new Standards Representative. Dan's letter is on the next page. If you want to talk to Dan about standards, you might want to order a copy of the proposed ANSI APL standard from the Association for Computing Machinery. To order the standard, write ACM Order Department, P O Box 64145, Baltimore, MD 21264. The cost is $18.75 for ACM members and $25 for non-members.

COMING NEXT ISSUE . . .

The Silent Majority will speak out on what it feels like to have a strong DEC commitment. The Raucous Minority will discuss wierd tricks to make public domain software do useful work. Big Boat People will explain how sail (sink?) majestically into the sunset. If you recognize yourself as a member of one of these socio-economic groups before 15 August 1985, please send your articles to:
Doug Bohrer
903 Ridge Road, Suite 3
Wilmette, IL 60091 USA

HELP WANTED - JUNTA

Douglas Bohrer
Editor, The Special Character Set

The APL SIG Steering Committee still is looking for someone to fill an opening on the APL junta. We are prepared to be flexible on your duties. We need some help covering the APL suite and attending DECUS standing committee meetings at symposia. If you plan to attend at least one symposium in the next year and would like to help out, write the editor of The Special Character Set or call a member of the Steering Committee.

APL SIG OPERATING PROCEDURES

Douglas Bohrer
Editor, The Special Character Set

The APL SIG Steering Committee has decided to use the following operating procedures to fill openings in the Steering Committee.

1. The APL SIG Steering Committee will consist of five members. The Steering Committee will assign appropriate duties to each of the five members and elect the SIG Chairman.

2. Steering Committee members are elected for life. Vacancies are created only by:
   A. Resignation
   B. Removal of a Steering Committee member by a majority vote of the Steering Committee.

3. Vacancies in the Steering Committee will be filled within eighteen months of the time the vacancy occurs. The vacancy will be announced in the APL SIG Newsletter as soon as possible.

4. Any member of the APL SIG may volunteer to fill a vacant Steering Committee position. A volunteer must send a letter announcing his candidacy to the editor of the SIG Newsletter before the publication deadline of the issue in which ballots are to be printed.

5. If there is only one volunteer for each open position, the Steering Committee may declare the position(s) filled without holding an election.

6. If there are more volunteers than open positions, an election must be held. Ballots will be printed in the APL SIG Newsletter. To vote, members must return their marked ballots to the DECUS office. Ballots must have a valid DECUS membership number and be signed. The DECUS office will count the votes and notify the Steering Committee of the results.

TOUCH TECHNOLOGIES
609 South Escondido Blvd., Suite 101 • Escondido, California 92025 • (619) 743-0494

8 March, 1985

Doug Bohrer
903 Ridge Ste, 3
Wilmette, IL 60091

To whom it may concern:

I am delighted to volunteer my time as the Standards Representative for APL Standards Committee.

I have had a long interest in APL having worked on an APL compiler in High School.

I look forward to serving on the Board.

Sincerely,

Dan Esbensen
President

DE/df

31 January 1985

Doug Bohrer
903 Ridge Rd.
Wilmette, Ill 60091

Dear Doug,

VAX APL Version 1.3 is the start of something very exciting.

Through the years the predominant marketplace for the APL language has been the financial and insurance institutions. In fact, one recent study, the LOMA Survey of May 1984, which represents over 100 of the major insurance companies of North America, reported that approximately 84 percent of all respondees indicated that APL was the single-most important language in use by their actuarial departments. We have also seen a slow, but steady increase in the number of users from the technical and engineering environments.

Although many of the enhancements planned for VAX APL and its product set which make up this new aggressive posture are not available in this maintenance update, VAX APL Version 1.3 begins the process by building a product set with its foundation based upon a variety of newly supported VAX processors. VAX APL Version 1.3 supports MicroVAX I (D-Floating), VAXstation I, and the 8600. Combined with our traditional set of processors these new entries provide our sales force with cost/performance solutions to many different customer environments. The strength of our product continues to be reinforced by the fact that customer application software and program compatibility amongst all of our VAX processors is guaranteed, thus providing the customer with upward and downward computing potential. As with VAX APL Version 1.2, VAX APL Version 1.3 is 60 percent faster than Version 1.0. This is accomplished through a number of optimizations made to specific primitives and in combination with general interpreter speed-ups.

For a detailed description of this announced release please review VAX APL V1.3 SPD, AE-P450E-TE. If you need further information on the VAX APL Product Set or are in need of additional support please feel free to give me a call. We are very confident that this year we will provide the types of enhancements, tools, and product offerings which will allow our sales force to sell a variety of computing solutions to our current and future VAX APL customers.

This is the year that Digital is to become a pace setter in the APL marketplace.

By the end of the year we will be considered a serious and committed APL vendor.

Dave Quigley
VAX APL Product Manager
603-881-2343

VAX APL OVERVIEW FROM FALL '84 DECUS

Stan Whitlock
APL Development Project Leader
DEC ZKO2-3/N30
110 Spit Brook Road
Nashua, NH 03062

The following article summarizes the "VAX APL Overview" session that was presented by DEC at Fall '84 DECUS in Anaheim. That session covered topics on VAX APL v1.2 (the current field image VAX APL product), APLSF-10/20 to VAX APL migration, and APL futures.

VAX APL v1.2-266 was made available to customers in July, 1984. It was a field-tested release that included many performance improvements. It represents a 40 to 60 % performance improvement in CPU time over VAX APL v1.0. The specific performance improvements fall into two categories, general speed-ups and specific optimizations of special cases. In the area of general speed-ups, the performance of the following features in VAX APL v1.2 has been improved:

- all monadic and dyadic scalar primitives
- indexed reference and indexed assignment
- faster floor algorithm in floor and near-integer testing
- relational functions on floating arguments
- monadic and dyadic transpose and rotate
- take, drop, compress, expand, and replicate
- catenate and laminate
- internal restructuring of the interpreter improves overall performance

Many examples of commonly used expressions were optimized with special case code, including:

- 1 1, 1 2, and 2 1 transpose of a matrix
- Boolean singleton compression of any array
- values to the first and second power
- catenate of any combination of a singleton and a vector
- plus, minus, floor, ceiling, and, and or reduction of a Boolean vector
- plus, minus, floor, and ceiling reduction of integer and floating vectors
- dyadic iota and epsilon on Boolean, integer, and character arguments
- "and dot equals" and "or dot not equals" inner products on a character array left argument and a character vector right argument
- and, or, less than, and less than or equals scan on Boolean vectors

APLSF-10/20 to VAX APL Migration

A package of tools called SFTOVX has been made available through the DECUS library that assists in transporting APLSF applications from TOPS-10 and TOPS-20 to VAX APL under VMS. The medium of transfer is ASCII text files. How these files are actually transferred from a TOPS-10/20 to a VAX/VMS system is outside the scope of this package: either ANSI labeled tape and DECnet could be used.

SFTOVX helps transport APLSF workspaces and files with /AS, /IS, /DA, and /BS organizations. Workspace variables, system variables, channel assignments, workspace identification, groups, and unlocked functions are transferred. SFTOVX is a series of APL workspaces, with functions that interact with the user to produce batch scripts that process APLSF data and then produce)INPUT driver files that process the APLSF data under VAX APL.

SFTOVX is available from the DECUS library as "20-176 SFTOVX: An APLSF to VAX APL Migration Utility".

There are a number of incompatibilities between APLSF-10/20 and VAX APL that affect the migration of APL applications from APLSF-10/20 to VAX APL. Some are hidden by SFTOVX, others are not:

- Escape mode mnemonics are not supported in VAX APL.
- Dyadic $ formatting is not in VAX APL; SFTOVX converts it to .BXFMT.
- Monadic .EN, .DE, and .OM are not in VAX APL; SFTOVX converts these to DEC-provided APL user-defined functions.

- A negative number raised to a non-integer power is an error in VAX APL.
- .FM output is slightly different in VAX APL.
- .IB is not in VAX APL; SFTOVX converts it to a DEC-provided user-defined function.
- .BXASCII is not in VAX APL and .BXAV is different in VAX APL; SFTOVX converts these to DEC-provided user-defined functions.
- Some APLSF system commands are not available in VAX APL:)BLOT,)CALL,)CREATE,)ECHO,)MODE,)MINCORE,)RUN,)SEAL, and)TABS are not in VAX APL.

A more complete description of incompatibilites between APLSF-10/20 and VAX APL is contained in Appendix C of the "VAX-11 APL Reference Manual", order number QL020-GZ (AA-P142B-TE), and in the documentation accompanying SFTOVX.

VAX APL Futures

The reader is asked to remember that "The following material should not be construed as a committement by Digital Equipment Corporation".

There are several areas for future development in VAX APL that are currently being considered:

- New APL primitive functions
- Integrated HELP facility
- Workspace analysis and debugging functions
- Continued performance improvements
- Special-case performance optimizations
- Calling external, non-APL routines as user functions
- Multi-key ISAM support
- Video editor support
- APL Workstation
- Access to VMS screen management primitives
- Run-time only APL

APL CHARACTER SETS FOR THE VT100 AND VT2

by Bob Awde Jr., Engineering Section Lea
General Mills, Inc.
9000 Plymouth Avenue North
Minneapolis, Minnesota 55427

For those of you that are using standarc DEC VT100 and VT200 terminals, it is possible to generate the APL character s on these terminals.

For the VT100, the Advanced Video Optior required to support alternate character sets. The procedure is as follows:

1. Generate the APL character on a 8316E ROM or 2716 PROM. Instructions for doing this are contained in the VT100 application note titled "How to Blast an Alternate Character ROM or PROM". This note can be obtained free of charge from DEC's Components group.
2. Install the PROM or ROM as noted in the application note. In our case, we elected to use 2716 PROM. Use of the 2716 requires one wire connection change. Again, the information needed to make this change is contained in the application note.
3. At this point, the VT100 should be able to generate APL character

For the VT200, the APL character set is down-line loaded into the VT200 by software. I have written a program that you can use to generate custom character set for the VT200 family of terminal. The program is called COMPOSE and can be ordered from the DECUS library (#11-760) An example set of files is included with the program that can be used to generate the APL character set.

The program itself is written to run on PDP-11 systems using either the RSX-11M or RSX-11M-PLUS operating systems. Howev the APL example file could probably be used with any operating system since typ this file on your VT200 terminal will generate and display the APL character s

For both the VT100 and VT200, the APL character set is selected by the Shift C (code 14 decimal) character and deselect by the Shift Out (code 14 decimal) chara cter. At our site, we have altered the A program to support the VT100 and VT200 terminals during the APL startup termina type selection phase.

There is one restriction that applies to both terminal types. The restriction is that the overstruck APL characters will displayed as one of the two characters i volved in the overstrike. Fortunately, y can enter overstruck characters in the n mal fashion and they can be printed out APL terminals such as the LA12 or LA120 with the APL character set option.

11-760

COMPOSE: VT200 Custom Character Set Generator Program

Version: V1.0, October 1984

Author: Bob Awde, Jr., General Mills, Minneapolis, MN

Operating System: RSX-11M V4.1, RSX-11M-PLUS V2.1

Source Language: FORTRAN 77, MACRO-11

Memory Required: 15,264 Words

Special Hardware Required: VT200 Family of Terminals

The COMPOSE program permits you to design and automatically generate custom character sets for the VT200 family of terminals. The output of COMPOSE consists of two files; a FORTRAN direct access file that contains the character definitions in binary form and a test file that can be "typed" at an appropriately configured VT200 terminal to actually create the custom character set. An example set of files used to generate the APL character set is included.

INSTRUCTIONS FOR USING THE COMPOSE PROGRAM

I. Program Description - The COMPOSE program permits you to design (compose) characters for the VT200 family of terminals. The output of COMPOSE consists of two files; a FORTRAN direct access file that contains the character definitions in binary form and a text file that can be "TYPED" at a VT200 terminal to actually create the custom character set. An example set of files is included that will generate the APL character set when "TYPED" at a VT200 terminal. Be sure that the terminal is in VT200 mode and that the terminal type is set to VT200 by using the RSX SET TERMINAL command. Note that the FORTRAN direct access file is APL.CHR and the test file that generates the character set is APL.TXT.

II. Invoke the compose program by typing: RUN COMPOSE

III. Enter a file name after the filename prompt. If the file does not exist, a new file with the name entered will be created. If the file already exists, the old file will be opened thus making it possible to modify existing custom character sets. This is the FORTRAN direct access file that will be used to store the custom character set. By convention, I use the .CHR file extension for these files.

IV. A menu containing four options will be displayed next. The first option permits you to create individual characters. The character to be composed is selected by entering in the COLUMN, ROW grid coordinates as pictured in the 7-Bit ASCII Code Table on page 2-3 of the VT220 Programmer Reference Manual.

After the character grid coordinates have been entered, either a blank character cell or the previously entered character cell is displayed depending whether the file is new or old respectively. At this point you create the character by moving the cursor around within the character grid using the four cursor control keys. To set (turn on) a pixel, press the numeric "1" key. To clear (turn off) a pixel, press the numeric "0" key. You may set and clear the same pixel as needed in order to develop your character.

After the character has been composed, type the letter "E" to exit and return to the main menu. At this point, you may compose the next character by reinvoking option 1 from the menu. You may re-select the same character to either verify its existence or make further modifications in the design of the character. Bear in mind that a completely created character set contains 95 characters.

V. Option 2 is used to create the text file that when typed at an appropriately configured VT200 terminal will both generate and display the custom character set. The file is given the extension .TXT. Bear in mind that entry into the custom character set is controlled by Shift Out (SO) and Shift In (SI) control characters. You must issue SO to enter the custom character set and SI to exit the custom character set.

VI. Option 3 permits you to create another custom character set. The program will prompt you for another file name as explained above. Repeat the steps explained above to create anotner custom character set.

VII. Option 4 permits you to exit the program after your custom characters set is completed.

Building APL-11 on RSX-11M/M-PLUS V4.1/2.1

By
Galen Tackett

I originally bought APL-11 from the DECUS library back in the days of 11M/M-PLUS Version 3.2/1.0 and installed it on an 11/70 running M 3.2. Until recently I had never rebuilt the APL task image since the original build; we just copied it from one system disk to another and it kept on working through 4.0 and 4.1, and the same task image runs okay on our new 11/44 with M+.

I decided to rebuild APL under M+ in order to gain a little workspace by using the supervisor-mode FCS resident library (FCSFSL); this required a little hacking on the ODL (overlay descriptor) file, but I'd done this kind of thing several times before and was quite confident of success.

Well, success wasn't quite what I got. Instead, when I ran it the rebuilt APL simply exited immediately without any kind of message. Suspecting some strange incompatibility with FCSFSL, I unhacked the ODL and built APL again, but without using FCSFSL. I was bothered, but not very surprised, to find that this didn't change anything.

It was time to bring out the big guns, which in this case meant getting a post mortem dump. I couldn't do this at first because it appeared the task was exiting "normally", but I soon discovered that under certain conditions (i.e., someone running something on another terminal) it would abort with a register dump. I soon had a crash dump file of about 1300 disk blocks. I printed about the first 100 pages of this, heaved a sigh, and sat down to pore over the octal dump for a while.

Before long the problem began to seem familiar: the task was aborting in the middle of trying to load an overlay. I'd had a similar problem with the DECUS C compiler when I built it (linked with FCSRES) on M 4.1, and I'd discovered then that including the right module from SYSLIB would cure that problem. The next step was obvious: I rebuilt APL (yet again), including the magic SYSLIB module (which, by the way, is called AUTO). Well, that was all it took: APL now worked just like it used to under M 3.2.

Next I returned to my original goal: to gain workspace by linking with FCSFSL. This was easy to do, and soon I had a working APL with a .bxwa of 32708. That's not a terribly big improvement over the 11M version with 32112, but it's still worth having.

I include below listings of APL7.CMD and APL7.ODL for RSX-11M-PLUS V2.1 and RSX-11M V4.1 (some simple editing is required for non-MPLUS or non-FCSFSL systems). These files are nearly identical to those supplied with APL-11; you will save time if you just edit my changes (clearly marked) into the original command files.

```
;
; APL7.CMD for RSX-11M V4.1 and RSX-11M-PLUS V2.1
;
; If your system doesn't support FCSFSL you must edit this file and the ODL
; file (see the comments in both places).
;
; You must also make a simple edit to APL7.ODL if you're using 11M instead of
; 11M-PLUS.
;
; The following line was changed to include the /nm and to refer to the
; FCSFSL version of the ODL file.
;
APL7/FP/CP/nm=APL7/MP
;
; APL-11 INTERPRETER TASK BUILD FILE
;
; APL-11 VERSION 02-00, RSX-11M VERSION 3.2
;                               RSX-11M-PLUS VERSION 1.0
;
; OPTION INPUT
;
TASK  =...APL
;
; SP STACK OF 256 WORDS
;
STACK =256
;
; APL-11 INTERPRETER LOGICAL UNIT NUMBERS
;      1-12     USER-AVAILABLE LUNS
;      13       UTILITY CHANNEL # 1
;      14       UTILITY CHANNEL # 2
;      15       USER'S TERMINAL
;
UNITS =15
ASG   =SY0:5:6:13:14,TI0:15
ACTFIL         =0
;
TSKV=SSTVT:8
;
; SINCE APL IS AN INTERPRETER, IT NEEDS A FLEXIBLE AMOUNT OF SPACE
; FOR MAXIMUM WORKSPACE SIZE, THE TOTAL TASK SPACE SHOULD BE AS
; CLOSE TO 32K WORDS AS POSSIBLE. FOR THIS, THE EXTENSION SIZE
; SHOULD BE PLAYED WITH TO DETERMINE OPTIMAL FOR EACH SYSTEM
;
EXTTSK         =4096
;
; FOR THE PURPOSES OF PATCH SPACE, WE WILL PROVIDE 400(8) = 256.
; WORDS OF PATCH SPACE
;
```

```
EXTSCT          =PATCH1:400
;
; The next line was added.  Comment it out if you aren't using FCSFSL.
;
suplib=fcsfsl:sv
/


;
; APL7.ODL for RSX-11M-PLUS V2.1 and RSX-11M V4.1
;
; For RSX-11M change all occurances of NOANSLIB to SYSLIB.
;
        .NAME   NULL,   NODSK
;
; THESE ARE THE APL USER READ-WRITE DATA AREAS
;
        .NAME   APLDF0, GBL
        .NAME   APLDF1, GBL
        .NAME   APLDF2, GBL
        .PSECT  DEF$0,  RW, D, GBL, OVR
        .PSECT  DEF$1,  RW, D, GBL, OVR
        .PSECT  DEF$2,  RW, D, GBL, OVR
;
; THESE SEGMENTS CONTAIN THE ERROR MESSAGE TEXT
;
        .NAME   ERSEG0, GBL
        .NAME   ERSEG1, GBL
        .NAME   ERSEG2, GBL
        .NAME   ERSEG3, GBL
        .NAME   ERSEG4, GBL
        .NAME   ERSEG5, GBL
        .PSECT  PATCH1, RO, I, GBL, OVR
        .PSECT  RSXTB0, RO, D, GBL
        .PSECT  RSXTB1, RO, D, GBL
        .PSECT  RSXTB2, RO, D, GBL
        .PSECT  RSXTB3, RO, D, GBL
        .PSECT  RSXTX0, RO, D, GBL
        .PSECT  RSXTX1, RO, D, GBL
        .PSECT  RSXTX2, RO, D, GBL
        .PSECT  RSXTX3, RO, D, GBL
        .PSECT  APLTB0, RO, D, GBL
        .PSECT  APLTB1, RO, D, GBL
        .PSECT  APLTX0, RO, D, GBL
        .PSECT  APLTX1, RO, D, GBL
;
; APL-11 V02-00 HAS A THREE-ROOT OVERLAY STRUCTURE, THE FIRST TWO
; CONSISTING OF CODE AND THE LAST CONSISTING OF REFRESHABLE
; (CLEAR) DATA
;
        .ROOT   BASE0,BASE1,BASE2
;
; The following line (BASE0) was modified
;
BASE0:          .FCTR   auto-01-lib-*(011,012,013,014,015,016,017,018,019)
;
; The next two lines (auto and lib) were added
```

```
;
auto:  .fctr   lb:[1,1]syslib/lb:auto
lib:   .fctr   lb:[1,1]noanslib/dl
O1:    .FCTR   O1A-O1B-PATCH1
O1A:   .FCTR   RT0-RT1-RT2-RT3-RT4-RT5-RT6-RT7
O1B:   .FCTR   RT8-RT9-RT10-RT11-RT12-RT13-RT14-APLDX7-DEF$O
RT0:   .FCTR   APLRX6/LB:ACOPY:APLEM:APLOP:APRINT:ARYSET:ASSIGN
RT1:   .FCTR   APLRX6/LB:BINSCH:BRKCHK:CARVE:CHRTAB:CLRCOR
RT2:   .FCTR   APLRX6/LB:CLRSI:CMDCHN:CMPRSS:CMPTRT:COMAND
RT3:   .FCTR   APLRX6/LB:COMRT:CONVER:DOT:DYAD:DYDEXP:DYDRT
RT4:   .FCTR   APLRX6/LB:ECHR:EMPTY:ERROR:EXEC:EXITIT:EXPR
RT5:   .FCTR   APLRX6/LB:FIXLIN:FNEXEC:FNIDEN:FRESET:FTOI
RT6:   .FCTR   APLRX6/LB:GETBLK:GETBYT:GETCOR:GETL:GETLIN
RT7:   .FCTR   APLRX6/LB:GETNB:GETNXT:GETREG:GETSAM:GETSCA:INIT
RT8:   .FCTR   APLRX6/LB:KILL:LEX:LIST:MONAD:MONRT:NAMOUT
RT9:   .FCTR   APLRX6/LB:NBRTRU:NEWLIN:NUMBER:NUMOUT:NXTIND
RT10:  .FCTR   APLRX6/LB:OCHAR:OPNSET:OUTBLK:OUTCHR:PKILL:PRINT
RT11:  .FCTR   APLRX6/LB:PRINTM:PUTL:READB:RESBLK:RESCHN:RESRT
RT12:  .FCTR   APLRX6/LB:SAME:SETBIG:SPASSN:STKXTN:SUBASN
RT13:  .FCTR   APLRX6/LB:SUBSCR:SYMBOL:TTINIT:TTOUT:VAL:WAITB
RT14:  .FCTR   APLRX6/LB:WRITEB
;
; OVERLAY STRUCTURE 1:  THIS CONTAINS, AMONG OTHER THINGS,
; THE .PARSE, .OPFNB, .CSI1, .CSI2, ETC...  CODE
;
;
; The next 11 lines (O11 thru O131) were modified or added.  If you aren't using
; FCSFSL, uncomment the first of these and comment out the second.
;
;O11:  .FCTR   APLRX6/LB:DECNAM:GETNAM-(O111,O112,O113)
O11:   .FCTR   APLRX6/LB:DECNAM:GETNAM
O111:  .FCTR   LB:[1,1]noanslib/LB:.CSI1:.CSI2
O112:  .FCTR   LB:[1,1]noanslib/LB:PARSE:.ODCVT-(O1121,O1122)
O1121:         .FCTR   LB:[1,1]noanslib/LB:PARSFN:PARSDV
O1122:         .FCTR   LB:[1,1]noanslib/LB:PARSDI:DIRFND
O113:  .FCTR   LB:[1,1]noanslib/LB:DLFNB:UDIREC:MRKDL
O12:   .FCTR   LB:[1,1]noanslib/LB:OPFNB-(O121,O122,O123)
O121:  .FCTR   LB:[1,1]noanslib/LB:ASSLUN:RETADR:CONTRL
O122:  .FCTR   LB:[1,1]noanslib/LB:RDWAIT:WATSET
O123:  .FCTR   LB:[1,1]noanslib/LB:CREATE:DEL:DIRECT:MKDL
O13:   .FCTR   APLRX6/LB:DATA-(O131,O132,O133,O134)
O131:  .FCTR   APLRX6/LB:CPIARG
O132:  .FCTR   APLRX6/LB:CPOARG
O133:  .FCTR   APLRX6/LB:SAVOVR:LODOVR
O134:  .FCTR   APLRX6/LB:DATIME:RAD:WSPRT
O14:   .FCTR   APLRX6/LB:FN
O15:   .FCTR   O15A-O15B
O15A:  .FCTR   APLRX6/LB:XERROR
O15B:  .FCTR   APLRX6/LB:ERRMSG-(O151,O152,O153,O154,O155,O156)
O151:  .FCTR   ERSEG0-APLTB0-APLTX0
O152:  .FCTR   ERSEG1-APLTB1-APLTX1
O153:  .FCTR   ERSEG2-RSXTB0-RSXTX0
O154:  .FCTR   ERSEG3-RSXTB1-RSXTX1
O155:  .FCTR   ERSEG4-RSXTB2-RSXTX2
O156:  .FCTR   ERSEG5-RSXTB3-RSXTX3
O16:   .FCTR   APLRX6/LB:APLOO1:SQRT
O17:   .FCTR   APLRX6/LB:APLOO2
```

```
018:   .FCTR    (018A,018B,018C,018D,018E,018F)
018A:  .FCTR    APLRX6/LB:APL003:MONIBM
018B:  .FCTR    APLRX6/LB:APL004
;
; The next 4 lines (018C thru 018C2) were modified or added.  If not using
; FCSFSL, uncomment the first line and comment out the second line.
;
;018C:          .FCTR    APLRX6/LB:APL005:PUTFA:GETFA-(018C1,018C2)
018C:  .FCTR    APLRX6/LB:APL005:PUTFA:GETFA
018C1:         .FCTR    LB:[1,1]noanslib/LB:PUTSQ
018C2:         .FCTR    LB:[1,1]noanslib/LB:GETSQ
018D:  .FCTR    APLRX6/LB:APL006
018E:  .FCTR    APLRX6/LB:APL007:CMDNBR:NGET
018F:  .FCTR    APLRX6/LB:APL008:SQRT
019:   .FCTR    019A-019B
019A:  .FCTR    APLRX6/LB:BLDLST:CLSFIL:ERASEC:GETLCL:GROUPC
019B:  .FCTR    APLRX6/LB:GRPC:OFF:SIV:SYMLST
;
; OVERLAY STRUCTURE 2:  THE MODULES IN THIS STRUCTURE ARE SMALLER
; THAN THOSE IN STRUCTURE 1
;
BASE1:         .FCTR    APLRX6/LB:ROOT1-*(020,021,022,023,024,025,026)
020:   .FCTR    020A-020B
020A:  .FCTR    APLRX6/LB:ASSFIL:CRTFIL:DROPC:RENFIL:RUNFIL:WSID
020B:  .FCTR    APLRX6/LB:LIBC
021:   .FCTR    APLRX6/LB:EXISTS:LODFIL:SAVFIL:STRTUP
022:   .FCTR    APLRX6/LB:COPYC
023:   .FCTR    APLRX6/LB:CKFNSI:FNLINE:LINKIL
024:   .FCTR    APLRX6/LB:FNHEAD:FNEND
025:   .FCTR    APLRX6/LB:ERASE:GLBSCH
026:   .FCTR    (027,028,029A,029B,029C)
;
; The following two lines were modified or added.  If you aren't using
; FCSFSL, uncomment the first line and comment out the second.
;
;027:  .FCTR    APLRX6/LB:TTYPE-LB:[1,1]noanslib/LB:FINIT
027:   .FCTR    APLRX6/LB:TTYPE
028:   .FCTR    APLRX6/LB:LISTER
029A:  .FCTR    APLRX6/LB:APL009
029B:  .FCTR    APLRX6/LB:APL010:ENDL:SCAMEM
029C:  .FCTR    APLRX6/LB:APL011
;
; THIS IS THE DATA OVERLAY STRUCTURE:  APL NEEDS TO BE ABLE TO
; REFRESH (I.E., RE-INITIALIZE) IT'S INTERNAL DATA AREAS WHENEVER
; THE USER DOES A )CLEAR COMMAND
;
BASE2:         .FCTR    APLDF1-DEF$1-(BASE2A,NULL)
BASE2A:        .FCTR    APLDF2-DEF$2
       .END
```

AMPL: A Modified Programming Language

Fred A. Masterson

Abstract

Increasing the total audience for APL and APL-like languages is desirable because of APL's many human-efficient features. AMPL is a modification of APL which is targeted for DECsystem-10 users who have been or would be discouraged by APL's non-standard keyboard and idiosyncratic internal editor.

Introduction

APL combines several features which enhance human efficiency:

1. Interactive, interpreted code.
2. Dynamic memory allocation.
3. Stored workspaces.
4. Simple syntax for procedure calls.
5. Powerful primatives for creating and altering data structures.
6. User access to system variables.

Because of these features, APL is one of the best widely available examples of a human-efficient programming language system. For this reason alone, it deserves a wider audience.

Unfortunately, many potential users of APL are discouraged by these additional features:

a. A special keyboard with unusual characters, and unusual locations for several standard characters.

b. An internal editor which differs markedly from common host computer editors.

It is not my purpose to debate the absolute merits or demerits of the last two features. The point I wish to make is that they put off a significant number of potential users. Interestingly, many of these individuals are sophisticated users of other computer languages and systems. Being conversant with other systems, they perceive the reliance on a special keyboard and editor as APL-chauvinism.

The use of single distinctive characters to name APL operators allows for a typographical conciseness unprecedented among most other programming languages. While some applaud this conciseness, others feel that it leads to unreadable code. My own conclusion is that APL's reputation as a "write only language" has resulted more from the styles of some APL programmers than from APL itself. However, there is a connection: APL's typographical economy, coupled with the conceptual economy of APL's operators, tempts programmers to push total economy to the extreme by writing highly compressed code. Another detriment to readable code is APL's tendency to ignore blank spaces which otherwise could be used to clarify the visual parsing of code into meaningful segments. Another difficulty is that the APL characters comprise a logographic notation as opposed to a phonetic notation, yet some people appear to be more comportable with phonetic notations [1]. Finally, the APL single character names lack mnemonic value. Some users - particularly beginners and occasional users - probably would appreciate the memory aid that would be provided by mnemonic names for the operators.

Independent of the pros and cons of special APL characters, the APL keyboard relocates such standard characters as colon, semi-colon, parenthesis, plus and minus. This poses a dilemma for users that must move back and forth between standard and APL keyboards. Recent research in information processing psychology has shown that significant attention must be focussed on a response selection task when the response rules keep changing [2]. By contrast, when the response rules remain constant for a long period of time, the response selection task becomes "automated", and the user does not need to allocate much attention to it. Particularly relevant here is the fact that the automatization process requires so much time that it will not occur if the user makes relatively frequent transitions between the two sets of response selection rules. Thus, users who frequently alternate between ASCII and APL keyboards will need to focus a significant amount of their attention on the response selection process. Since attention is a limited cognitive capacity, it follows that less attention will be available for such activities as planning a function definition.

Another problem is that of limited access to APL terminals. In my experience, terminals with the APL keyboard account for a small proportion of total terminals at academic computing centers. Hence, students with APL assignments must wait in line. It was for this reason that I discontinued teaching APL in my statistics courses, and undertook the project of modifying APL to allow standard characters and a standard editor.

To summarize thus far, many potential users who would be enthralled by APL's simplicity and power are nonetheless repelled by (to them) unreadable code, unappealing editors, and unavailable terminals. What follows is a description of a modification of APL which retains the obviously human efficient properties 1-6 (above) but eliminates the more controversial properties a-b (above). The goal was not to divest APL of properties that have endeared it to devoted users, but to

modify APL in order to increase its appeal for an alternative audience. I have dubbed the product AMPL: "a modified programming language". It runs on the Digital Equipment Corporation (DEC) DECsystem-10, is based on DEC's APLSF, and utilizes the DECsystem-10 SOS editor.

Implementation

DEC's APLSF allows the use of a standard keyboard if so desired. In some cases, a single standard character can be used in place of an APL character. Luckily, the representation of many common operators follows conventional algebraic notation (for example, addition is represented by '+' and subtraction by '-'.) Other functions and common operators are represented by single characters which are reasonably similar to those employed in conventional algebraic and computer notation (thus, multiplication is represented by '#', division by '%', and the value assignment operator by '←').

In other cases, DEC allows the users to substitute strings of standard characters for APL characters. As shown in Table 1, '.RO' may substitute for '⍴' (the rho or shape function) and '.DA' may be substituted for '↓' (the down-arrow or drop function). One problem with these substitutions is that they lack adequate mnemonic value. In addition, APLSF will not honor spaces between these operator names and the names of the surrounding arguments resulting in such monstrosities as 'N.ROSCORES' or '12.DASCORES' (where SCORES is a data vector or array). My solution is to give user-defined functions for these operations. In AMPL, the monadic from of '⍴' is called 'SIZE', and the dyadic form is named 'SHAPE'. Similarly, 'DROP' is the name of a function which performs the '↓' operation. These names have good mnemonic value. In addition, APL respects spaces on either side of the names of user-defined functions. Hence, the jumbled together examples give above are replaced by the more readable 'N SHAPE SCORES' and '12 DROP SCORES'.

These and other user defined functions are shown in Table 1. Some user defined functions have been created specifically for beginners. Thus, 'SUM', 'ROWSUM', and 'COLSUM' in Table 1 provide easy to understand alternatives to '+/' and '+/[1]'. The latter notations utilize APL's compression operator ('/') and can be reserved for more advanced users.

A possible criticism of this approach is that too much of the user's workspace is taken up with these user defined functions. However, the definitions are extremely brief. In addition, not all APL operators may need to be defined. Saal and Weiss [3] have shown that several APL operators are infrequently applied by the average user. Thus, a given user can customize a workspace to include user defined functions corresponding to just those APL operators that will be needed.

The current approach to using ASCII characters with APL is a simple one. It would be more satisfactory to lay an AMPL interpretor on top of the APL interpretor, or, better yet, directly modify the APL interpretor. Anyone interested in either enterprise should refer to Crick's [1] thoughtful article.

Let us turn now to the problem of using a standard editor with APL. I know of two systems which use the DECsystem-10 general purpose editor, SOS, for creating and modifying APL function definitions: one described by Orgass and Uhler [4] and one described by Masterson [5,6]. The current AMPL system is a combination of the two (see below).

The transfer of control between APL and SOS editor is completely transparent. The user merely invokes an APL function named EDIT, supplying as argument the name of the function to be created or modified. Thus, typing

```
EDIT 'MEAN'
```

automatically transports the user into SOS. Furthermore, if MEAN has been defined previously, the old definition is automatically loaded into the SOS work file. The user then can use standard SOS commands to modify the definition. When finished editing, the user types G which has the automatic effect of transporting the user back to APLSF along with the new definition (analogous to ∇ in standard APL). The net result is a smooth and simple flow between APLSF execution mode and an SOS-mediated function definition mode.

The implementation of the APLSF–SOS interaction consists of two main parts. The first is a procedure for initializing SOS so that it automatically edits the desired scratch disk file and so that the SPS "G" command will shift control back to APLSF. Masterson's procedure performs the initialization by first running SOS with the scratch file name, and invoking an option in the user's SWITCH.INI file which identifies the program to be executed when the user types SOS's "G" command (this program causes APLSF to be run with an address offset of 1). Orgass and Uhler's procedure is more sophisticated. In their system, the user runs an assembly language program named INITIA, which does all the required SOS initializations and then transfers control to APLSF.

TABLE 1

APL functions and operators expressed in APL characters, in APLSF equivalent standard characters, and as user defined functions in the present system.

Operation	APL Keyboard	Standard Keyboard	User Defined Functions
Remove the part of Y specified in X	X↓Y	X.DAY	X DROP Y
Take the part of Y specified in X	X↑Y	X^Y	X TAKE Y
Find the dimensions of Y	⍴Y	.ROY	SIZE Y
Restructure Y according to the dimensions specified in X	X⍴Y	X.ROY	X SHAPE Y
Generate a vector of the first N integers	⍳N	.ION	GENINTS N
Get input from keyboard	⍞	.BX	KEYB
Base 10 logarithm of y	10⍟Y	10.LGY	LOG Y
Transpose array Y	⍉Y	.TRY	TRANSP Y
Invert matrix Y	⌹Y	.DQY	INVERT Y
Execute string S as APL code	⍎S	.XQS	EVAL S
Sum across a vector Y	+/Y	+/Y	SUM Y
Sum across the rows of a matrix Y	+/Y	+/Y	ROWSUM Y
Sum across the columns of a matrix Y	+/[1]Y	+/[1]Y	COLSUM Y

The second part of the APLSF–SOS interaction is an APL editing function which sends control to SOS. If the definition to be edited already exists, the editing function must write the current definition onto the SOS scratch disk file. Upon return from SOS, the editing function must read the new or modified definition from the scratch disk file. Orgass and Uhler's edit function (named SOS) allows the user to edit several APL definitions in one pass through SOS, while Masterson's does not. However, Masterson's edit function (named EDIT) has the advantage of containing several traps to catch errors which could abort execution and, in some cases, prevent recovery of the edited definition.

The current AMPL system combines the Orgass and Uhler INITIA program with Masterson's EDIT function. As already mentioned, the system contains several user-defined functions with mnemonic names to perform primative APL operations. A complete description is available on request [7].

Discussion

My goal was to separate the human efficient features of APL from the APL special character set and the APL editor. The result, AMPL, is a user- friendly high level programming language that can be appreciated from several perspectives.

The ability to use SOS in APLSF is appreciated by computer novices and advanced users alike. Novices appreciate the fact that the editor learned in conjunction with APL can serve them equally well in other DEC-10 applications, such as formatting a term paper or composing a PASCAL program. Advanced DECsystem-10 users who already know SOS appreciate the convenience of using SOS with APL.

Users of a standard keyboard for other applications have remarked favorably on the use of a standard keyboard for AMPL. Indeed, it is a real pleasure to go from some other standard keyboard application to AMPL and not experience the flurry of typographical errors that usually occurs on transition to the APL keyboard.

The human efficiency of AMPL's mnemonic names with standard characters versus APL's special characters depends on the background of the user. Beginners find the mnemonic names easier to learn and use. Experienced but occasional users also may benefit from the memory aid provided by mnemonic function and operator names. By contrast, intensive users who do most or all of their computing with APL may prefer the typographical economy provided by the APL special character operator names. Thus, as mentioned earlier, AMPL is not a substitute for APL, but a repackaging for a different user audience.

Part of the motivation behind AMPL has been political. Widening APL's audience will, I hope, raise popular standards regarding the acceptability of high level programming languages. Many users are unaware of APL's human-efficient features outlined at the beginning of this report, and hence unquestioningly accept "new" languages which lack these features. Were such a user's consciousness raised by exposure to APL or AMPL, he or she would demand more human efficiency from new languages.

Few would argue that APL is a perfect example of a human-engineered programming language. For example, an optional compilor would be desirable, as would the inclusion of structured flow of control. However, APL appears to be the best widely available example of a human-efficient programming language system – and AMPL seems to be a good way to present APL to many potential new users.

References

[1] Crick, M.F.C., *An ASCII notation for APL.* Quote Quad 11 (1980), 18-25.

[2] Shiffron, R.M. and Schneider, W., *Controlled and automatic human information processing: Perceptual learning, automatic attending, and a general theory.* Psychological Review 84 (1977), 127-190.

[3] Saal, H.J. and Weiss, Z., *An empirical study of APL programs.* Computer Languages 2 (1977), 47-59.

[4] Orgass, R.J. and Uhler, G.M., *Using SOS in APLSF.* Technical Memorandum No. APLAD13, September 14, 1977, Department of Computer Science, University of Arizona, Tucson, Arizona 85721.

[5] Masterson, F.A., *Beginner's introduction to APL on the DEC-10.* Technical Memorandum No. AMPL 1, August 15, 1978, Software Psychology Project, Department of Psychology, University of Delaware, Newark, Delaware 19711.

[6] Masterson, F.A., *Bringing APL down to earth on the DECsystem-10: Standard characters and a standard editor.* Behavioral Research Methods and Instrumentation (1981), to appear.

[7] Masterson, F.A., *DEC-10 AMPL installation guide.* Technical Memorandum No. AMPL 3, August 15, 1981, Software Psychology Project, Department of Psychology, University of Delaware, Newark, Delaware 19711.

Software Psychology Project
Dept. of Psychology
University of Delaware
Newark, DE 19716
U.S.A.

[Editor's Note: This article originally appeared in APL Quote Quad, Volume 15, Number 2, December, 1984, pp. 17-20]

NEWSLETTER SUBSCRIPTIONS AND OVERLAP AS OF DECEMBER, 1984

	APL	BAS	DIC	DTR	EDU	IAS	LGS	LHS	LTS	MMP	NTW	OAG	RST	RSX	RT	UNI	VAX
AP	612																
BA	547	1792															
DI	453	725	1109														
DT	520	1086	754	2058													
ED	500	738	532	677	1024												
IA	438	445	420	458	464	588											
LG	452	523	474	549	523	420	792										
LH	499	754	508	832	613	468	526	1635									
LT	492	764	594	940	573	455	543	929	2006								
MM	428	469	428	463	442	414	429	492	483	634							
NT	489	937	630	1191	658	502	646	1033	1234	498	2469						
OA	468	824	688	1041	615	438	546	867	1029	481	1158	1995					
RS	458	919	642	686	623	427	468	530	559	430	634	607	1220				
RX	507	814	557	1005	550	482	519	941	989	481	1319	906	601	2531			
RT	501	735	590	649	534	461	478	837	768	472	750	718	571	987	1968		
UN	453	528	469	570	503	434	488	617	826	446	816	679	485	683	610	1137	
VA	562	1290	866	1756	839	515	709	1295	1667	562	2260	1509	801	1723	1040	997	4741

NOTE: Above numbers contain 404 subscribers who receive ALL newsletters

15 April 1985

Dear APL Fanatic:

I want to thank all of you who wrote something for this issue. This level of support is great for keeping the APL SIG alive and your editor mentally healthy. In recent DECUS meetings the powers that be have expressed confidence in the continued viability of this publication. I think this is a tribute to your support.

However, before everyone relaxes, please remember that the saga continues. If the next issue doesn't have enough material the Force will no longer be with us. Please write something for the next issue and send it in.

Almost anyone can write an article. It doesn't have to be long or heavy. Write an article about how your site uses APL or/and how it chose APL in the first place. Write an article about that handy function you just wrote. Write a testimonial about how APL changed your life.

Maybe you already have an article on an APL topic, but you published it somewhere else. Let me republish it if you can give me copyright permission.

I have printed a style sheet for any article you write. However, even if your article is in another format I'll take it. I can read RT-11 RX02 floppies. I have a volunteer typist. JUST GIVE ME SOMETHING TO PRINT!

The deadline for the next issue is 15 August 1985. However, if you have something now SEND IT NOW. I have only 32 pages to fill and it's first in, first out. (With a bluff like that I expect you will all ask to play poker with me as soon as possible.)

Sincerely,

Doug Bohrer
Bohrer and Company
903 Ridge Road, Suite 3
Wilmette, IL 60091 USA
Phone: 312-251-9449

Printed in the U.S.A.