THE SPECIAL CHARACTER SET

This is the FIRST Subscription Service Issue of the APL SIG Newsletter

APL SIG

DECUS

No. 6

Printed in the U.S.A.

"FROM HIEROGLYPHICS TO APL
THE RELENTLESS COURSE OF HISTORY"

Homer Hartung, Senior Scientist
Philip Morris, USA
Research Center
P. O. Box 26583
Richmond, Virginia 23261

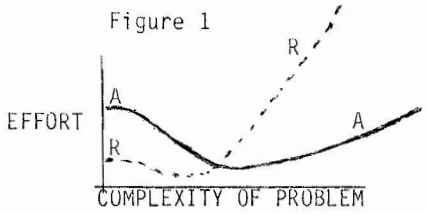
Records of ancient kingdoms and their rulers are found in hieroglyphics. Some of these gave numbers for populations, armies, herds, etc. The schemes used by the ancient scribes were very simple. In ancient Egypt only a small number of symbols were needed to represent all numbers up to very large counts.

Have you ever wondered why we don't use the simpler hieroglyphic representations? Probably not, because we are taught elementary mathematics before understanding the source of its symbols and procedures. Thus, we are indoctrinated with the idea that our representations of numbers were somehow decreed by God. Continued use of Roman numerals on cornerstones, monuments and the like, however, testifies to the contrary.

If you stop and think about it, there would be some advantages to going back to wider use of the Roman numerals. We could do away with ten of the top row of keys on typewriters and terminals, and it would be a lot easier to teach to young children for simple counting tasks. You might worry that advanced mathematics would be impossible, but that isn't necessarily the case if we are smart enough. Consider old man Euclid, who was pretty swift in geometry even without modern computing techniques---but then he was a real genius!

The real pros and cons of number representations can be illustrated graphically as in Figure 1.

Figure 1



This shows a schematic representation of effort expanded versus complexity of the problem. 'R' stands for Roman numerals and 'A' for Arabic. In both cases, there is an initial hump associated with the difficulty of starting out. The hump for the 'A' curve is larger because the Arabic numbers are based on ten new symbols and rigid rules on evaluation by relative placement. The hump for the 'R' curve is smaller because it involves simple association of counts with familiar letters. The 'R' line is below the 'A' line when problems have a low degree of complexity. Tallies of ballots, for example, are certainly much easier with an 'R' type system than with Arabic numerals. There is a crossover point and a rapid divergence for increasing complexity, however. It is hard to conceive of trying to work logarithms using Roman numerals!

Editorial

The history of number systems provides a strong parallel with things that are now happening in the area of computer programming languages. The concept of programming languages is new with our present generation. The first one was FORTRAN, invented around 1953, and thousands of similar approaches have been tried since then. A radical departure from the FORTRAN-type of language was proposed in a book entitled "A Programming Language" by Kenneth Iverson in 1962. This introduced a notational scheme which was an extension of matrix algebra. It was given the abbreviation APL and introduced as a computer language in 1969.

Except for APL, all computer languages invented to date have been ad hoc adaptations of mathematical notation such that it can be displayed easily with regular typewriter symbols. The assumption is that the standard typewriter keyboard was also devised by God! All such languages are analogous to the situation with Roman numerals 2000 years ago--only familiar symbols are used.

APL is now in same situation as Arabic numerals when they were first introduced to Westerners--new and unfamiliar symbols are required. Thus, Figure 1 also represents the pros and cons of APL versus other languages. The 'A' curve represents relative effort in using APL to solve computational problems while 'R' represents what we might call 'regular' programming languages with a standard keyboard.

The parallel extends to the fact that whatever we first learn tends to seem easier and better. Children are not conscious of the hump in the effort curve for using Arabic numerals because they do not know enough to question the rules of the game. Similarly, people who start out learning APL as their first computer language don't have much trouble.

The history of mathematics reveals that the Arabic numerals were around for many centuries before they became accepted widely. Obviously when hieroglyphics are taught to children for counting it is tough to switch them over to a superior but more complex scheme. By the same token, we can expect that the acceptance of APL as a widely used programming language will come about gradually but relentlessly as people make increasingly complex demands on computer systems.

Imagine the fate of an accountant who might try to keep books in Roman numerals because they are easier to learn for hand tallies of inventory. The same fate awaits programmers who insist on using regular languages because they are easier for simple computing tasks.

(These thoughts were picked up at APL83, particularly from Prof. D. B. McIntyre.)

EXTRA EXTRA...

An EXTRA issue of The Special Character Set will appear if the Editor receives enough articles for it by December 1, 1983. If Thanksgiving overeating keeps you from meeting the EXTRA deadline, the next regular issue will be open for submissions until February 15, 1984. Send your article, letter or postcard to:

Douglas Bohrer
Bohrer & Company
903 Ridge Road, Suite 3
Wilmette, IL 60091 USA

THE SPECIAL CHARACTER SET
September 1, 1983 No.6

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

For subscription information and an application, contact:

Membership Services
DECUS
One Iron Way
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.

CONTENTS

Local Variables

ROLL WITH APL IN LAS VEGAS

Susan M. Abercrombie
APL SIG Symposium Coordinator
Ventrex Laboratories, Inc.
Portland, ME 04103

For Las Vegas, October 24-28, the APL SIG is sponsoring sessions for both PDP-11 users and those with larger systems.

DIGITAL will present two sessions covering DEC's APL products for the 10/20 and VAX machines.

A tutorial will introduce the features of APL and programming techniques to new or potential users. A follow-up session will show how sophisticated report formatting can be simply and easily done in APL.

A presentation especially for RSTS users will describe improvements to APL-11 version 1 on RSTS [available from the DECUS library, #11-SP-43 in the new catalog]. A user panel will provide a forum for discussion of problems and solutions concerning APL-11 in any of its forms (V1 on RSTS, V2 on RSX [DECUS library #11-SP-25], V1 or 2 on RT [DECUS library #11-476 for V2 or check the RT SIG symposium tapes for V1).

We will be sharing a suite with the MUMPS SIG (strange bedfellows?). Check the area near the info booth for the room number. We hope to have a terminal available for the tutorial and during limited hours in the suite. These facilities should make it much easier for those who are intrigued to gain an appreciation for this unique language.

Scheduled session times are:

AN APL TUTORIAL	Mon 12:30- 2:00
FORMATTING REPORTS IN APL	Tue 11:00-12:00
APL INTERNALS FOR DIGITAL SYSTEMS	Tue 2:00- 3:00
APL FOR TOPS-20 AND VAX/VMS	Tue 3:30- 4:00
RSTS 'SYSTEM CALLS' IN APL	Wed 8:30-10:00
WRITING A PDP-11 APL IN C	Thu 8:30- 9:30
APL-11 USER PANEL	Thu 5:00- 6:00

Consider also the following session, sponsored by RSTS:

HOW TO MAKE IBM COBOL READABLE TAPES ON RSTS (using APL and BASIC-PLUS)	Thu 12:00- 1:30

See you in Las Vegas!

HELP WANTED: APL SIG STEERING COMMITTEE

Douglas Bohrer
Editor, The Special Character Set

This is your chance to become a member of the powerful ruling junta of the APL SIG. Due to the resignation of William Kaufman, there is an opening on the APL SIG Steering Committee. Since Steering Committee members have emergency powers for life, this may be your only chance to become one of the "big guns" in the APL SIG. Your duties will be your choice of either Symposia Coordinator or Library Coordinator.

If you choose Symposia Coordinator, you get to beat the bushes for speakers, beat speakers into meeting submission deadlines and fight scheduling battles over what sessions are scheduled against APL. You will need to attend both Fall and Spring U.S. Symposia at your company's expense and be available for 2 possible scheduling trips to Marlboro at DECUS expense. At the Symposia, you will get up at an ungodly hour each morning to eat breakfast and talk shop with the Symposia Committee.

If you choose Library Coordinator, you get to figure out how to coordinate library submissions from at least 3 different machines and 6 different operating systems with incompatable media and interpreters. In addition you become our representative to the Library Committee. This may involve one trip a year at DECUS expense to meet with them as well as Library meetings at Symposia. Hopefully, you will attend at least one U.S. Symposium a year.

To enter, simply write the Editor of this newsletter a letter explaining your candidacy. You need not chose whether you want to be Symposia or Library Coordinator until after you are elected. The Special Character Set, No. 4, October 15, 1982 has the letter that our current Chairman, Larry LeBlanc, wrote. To be valid your letter must reach the Editor before February 15, 1982. Send your letter to:

Douglas Bohrer
903 Ridge Rd Suite 3
Wilmette, IL 60091

The APL SIG Operating Procedures are explained in full below.

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.

State Indicators

NEW LANGUAGES COMPETE WITH APL

A Usually Reliable Source
Digital Equipment Corporation
Somewhere in New England

APL, BASIC, FORTRAN, COBOL ... these programming languages are well known and (more or less) well loved throughout the computer industry. There are numerous other languages, however, that are less well known yet still have ardent devotees. In fact, these little-known languages generally have the most fanatic admirers. For those who wish to know more about these obscure languages -- and why they are obscure -- I present the following catalogue.

SIMPLE

SIMPLE is an acronym for Sheer Idiot's Monopurpose Programming Linguistic Environment. This language, developed at Hanover College for Technological Misfits, was designed to make it imposible to write code with errors in it. The statements are, therefore, confined to BEGIN, END, and STOP. No matter how you arrange the statements, you can't make a syntax error.

Programs written in SIMPLE do nothing useful. Thus they achieve the results of programs written in other languages without the tedious, frustrating process of testing and debugging.

SLOBOL

SLOBOL is best known for the speed, or the lack of it, of its compiler. Although many compilers allow you to take a coffee break while they compile, SLOBOL compilers allow you to travel to Bolivia to pick the coffee. Forty-three programmers are known to have died of boredom sitting at their terminals while waiting for a SLOBOL program to compile.

VALGOL

From its modest beginnings in Southern California's San Fernando Valley, VALGOL is enjoying a dramatic surge of popularity across the industry.

VALGOL commands include REALLY, LIKE, WELL and Y*KNOW. Variables are assigned with the =LIKE and =TOTALLY operators. Other operators include the California Booleans, FERSURE and NOWAY. Repetitions of code are handled in FOR - SURE loops. Here is a sample VALGOL program:

```
LIKE, Y*KNOW (I MEAN) START
IF PIZZA =LIKE BITCHEN AND
B =LIKE TUBULAR AND
C =LIKE GRODY**MAX
(FERSURE)**2
THEN
FOR I =LIKE 1 TO OH MAYBE 100
DO WAH - (DITTY**2)
BARF(I) =TOTALLY GROSS(OUT)
SURE
LIKE BAG THIS PROGRAM
REALLY
LIKE TOTALLY (Y*KNOW)
```

VALGOL is characterized by its unfriendly error messages. For example, when the user makes a syntax error, the interpreter displays the message: GAG ME WITH A SPOON!

LAIDBACK

Historically, VALGOL os a derivative of LAIDBACK, which was developed at the (now defunct) Marin County Center for T'ai Chi, Mellowness and Computer Programming, as an alternative to the more intense atmosphere in nearby Silicon Valley.

The center was ideal for programmers who liked to soak in hot tubs while they worked. Unfortunately, few programmers could survive there for long, since the center outlawed pizza and RC Cola in favor of bean curd and Perrier.

Many mourn the demise of LAIDBACK because of its reputation as a gentle and nonthreatening language. For example, LAIDBACK responded to syntax errors with the message, SORRY MAN, I CAN'T DEAL BEHIND THAT.

SARTRE

Named after the late existential philosopher, SARTRE is an extremely unstructured language. Statements in SARTRE have no purpose; they just are. Thus SARTRE programs are left to define their own functions. SARTRE programmers tend to be boring and depressed and are no fun at parties.

FIFTH

FIFTH is a precision mathematical language in which the data types refer to quantity. The data types range from CC, OUNCE, SHOT and JIGGER to FIFTH (hence the name of the language), LITER, MAGNUM and BLOTTO. Commands refer to ingredients such as CHABLIS, CHARDONNAY, CABERNET, GIN, VERMOUTH, VODKA, SCOTCH, BOURBON, CANADIAN and WHATEVERSAROUND.

The many versions of the FIFTH language reflect the sophistication and financial status of its users. Commands in the ELITE dialect include VSOP, LAFITE and WAITER'S(RECOMMENDATION). The GUTTER dialect instead has commands for THUNDERBIRD, RIPPLE and HOUSE(RED). The GUTTER dialect is a particular favorite of frustrated FORTH programmers who end up using this language.

C-

This language was named for the grade received by its creator when he submitted it as a class project in a graduate programming class. C- is best described as a "low level" programming language. In general, the language requires more C- statements than machine-code instructions to execute a given task. In this respect it is very similar to COBOL.

Global Variables

LITHP

This otherwise unremarkable language is distinguished by the absence of an "S" in its character set. Programmers and users must substitute "TH". LITHP is said to be usefull in prothething liththth.

DOGO

Developed at the Massachusetts Institute of Obedience Training, DOGO heralds a new era of computer-literate pets. DOGO commands include SIT, STAY, HEEL and ROLL OVER. An innovative feature of DOGO is "puppy graphics," a small cocker spaniel that ocassionally leaves deposits as he travels across the screen.

RENE

Named after the famous French philosopher and mathmetician Rene DesCartes, RENE is a language used for artificial intellegence. The language is being developed at the Chicago Center of Machine Politics and Programming under a grant from the Jane Byrne Victory Fund. A spokesman described the language as "just as great as dis [sic] great city of ours." The Center is very pleased with progress to date. They say they have almost succeeded in getting the machine to think. However, sources inside the organization say that each time the machine fails to think it disappears.

[Editor's Note: This internal DEC study was leaked to me without a copyright notice. The last item was added by The Special Character Set correspondent in Chicago.]

FREE NEWSLETTERS IN EUROPE

Ken Litsios
Activities Coordinator
DECUS Europe
P.O.Box 510
CH-1213 Petit-Lancy 1
Geneva, Switzerland

You will be pleased to hear that Digital Europe has agreed to pay for all the U.S. SIG newsletters for European subscribers for this coming fiscal year. Therefore, until July 1984, you will continue to receive the SIG newsletters free of charge. After this date it is most likely that you, like the U.S. members, will have to pay to receive the SIG newsletters. During the coming year, your Chapter Board will be discussing alternative ways to fund this service for it's members. I am sure that they would welcome your inputs on this matter.

[Editor's note: This item is quoted from a letter from Mr. Litsios to Dipl. Ing. Ralf Herzer, Forshungsinstitut der DBP beim FTZ, Postfach 42 02 00, 1 Berlin 42. From the copy Mr. Litsios sent me, it seems that Mr. Herzer is interested in contacting other European APL users in DECUS.]

FANTASTIC PIZZA OFFER: FIXING APL-11

Ariel Cohen
The Cognitive Laboratory
Haifa University
Derech Aba Huchi
Haifa 31999
Israel

It was surprising to read that Digital has not even discussed a new APL for the PDP-11 over pizza, since, as far as I know, they continue developing new machines using the PDP-11 instruction set. Anyway, there seems to be a good market for a good implementation. I even agree to send Digital's staff pizzas from the same restaurant which baked the pizza they were eating while discussing the new VAX APL with a 4 gigabyte workspace...

Fixing APL for RSX

About a year and a half ago I ordered Digital's APL for RSX on a DECUS tape. Knowing that it is a Digital product, I was disappointed to receive a SYSTEM ERROR while trying APL V2.1 for the first time! I have fixed some bugs which I encountered (and a few documented problems). Since then I have used APL V2.1, but only for small projects. It is still too risky.

I have encountered more bugs, which I did not have time to correct. In July and August I am going on vacation and I intend to use part of the time for working on the interpreter. The only problem which discourages me is that the maximum workspace is 32KB and enlarging it would mean a great deal of work. On the other hand, I enjoy the ironic jokes which fill the comments in the sources.

If you know of any bugs in V2, please inform me about them.

[Editor's note: I immediately informed Roger Matus, DEC's APL Product Manager and former APL SIG Chairman, of this fantastic offer of free pizza. His reaction came in two parts. First he said that the APL engineering staff was engulfed in other projects and would never consider a new APL-11 because the market seemed too limited. Then he said that the current fashion at DEC is Chinese food. My immediate offers of stir fried meals did not seem to change his mind on APL-11.]

NEWSLETTER BACK ISSUES AVAILABLE

Back issues 1-5 of The Special Character Set are available from the DECUS Library for $5(US). The program order number is "APL Newsletters." To order, send a completed DECUS Library form and your $5 to:
DECUS Order Processing
One Iron Way/MRO2-1/C11
Marlboro, MA 07152

Back issues will also be sold in the DECUS Store at the Fall 1983 US Symposium in Las Vegas.

Functions and Idioms

VT100 FULL SCREEN EDITING USING ⎕ARBIN

Robert L. McGhee
VAX System Programmer
STSC, Inc.
2115 East Jefferson Street
Rockville, Maryland 20852
(301) 984 5327

Introduction

Frequently in APL applications there is need for the user to be able to edit text. One such application is the development of screens of text for computer-aided instruction courses. The VT100 is an attractive terminal for these applications where the APL character set is not required. This article describes a technique for implementing a full-screen editor in just a few lines of APL code. The technique and code were developed by Mr. Ed Myers of STSC, Inc. This approach can be implemented on any APL system that provides a facility for receiving each character as it is entered from the keyboard and transmitting any of the 128 ASCII characters to the terminal. On STSC's APL*PLUS/2000 System this facility is the ⎕ARBIN system function.

The display screen is considered to be an APL character matrix with, say, 24 rows and 80 columns. To edit the screen, the user moves the cursor to the desired location and deletes or adds characters at that point in the text. At the end of the editing session, the character matrix in the workspace agrees exactly with the display screen. The following explanation begins with the lowest level function, ⎕ARBIN, that provides the interface between the APL workspace and the terminal. At the next level up, the INKEY function uses ⎕ARBIN to position the cursor on the screen and receive characters from the keyboard. At the highest level, the EDITSCREEN function displays the character matrix to be edited and returns an edited version of this matrix as its result.

⎕ARBIN: APL-VMS Interface

The ⎕ARBIN system function uses the VMS $QIO system service to transmit its right argument to the terminal. The left argument specifies the number of characters to be received from the terminal. When this count is reached, the characters received are returned as the explicit result.

The data type of both the right argument and the explicit result is numeric; the internal representation is one-byte integer. The values correspond to ASCII codes; for example, 27 corresponds to ESC the escape character and 65 corresponds to A. The "passall" mode is used with $QIO to bypass VMS interpretation of control characters and escape sequences. This means that all control characters and escape sequences are passed directly into the APL workspace where they can be interpreted by an APL function.

```
    ∇ R←CHAR INKEY LOC;⎕IO;A
[1]   ⎕IO←0
[2]   TOP:R←1 ⎕ARBIN CHAR,XTBL[LOC[0];],YTBL[LOC[1];]
      ∨ →(R∈ 27 127)↓0
[3]   →(R=127)/L2 ⍝ DEL
[4]   A←2 ⎕ARBIN⍳0 ∨ R←A[1]- 61 80 0[91 79 ⍳A[0]] ◇
      →0 ⍝ ESC
[5]   L2:R←8 ⍝ DEL
    ∇
```

```
    ∇ CURSOR LOC;⎕IO
[1]   ⎕IO←0 ∨ ⎕ARBOUT XTBL[LOC[0];],YTBL[LOC[1];]
    ∇
```

INKEY: Terminal I/O Control

The INKEY function is dyadic and returns an explicit result. The argument CHAR is displayed at the current cursor location; the caller of INKEY is responsible for positioning the cursor ahead of time. Next the cursor is repositioned to the address specified by LOC. At this point the ⎕ARBIN function looks for a character from the terminal and when one is available returns it in the result R.

Two characters receive special treatment: the DEL (127) delete character and the ESC (27) escape character. If a delete character is entered, it is converted into a BS (8) backspace character; the calling function implements "destructive backspace" so BS performs the delete operation.

If an escape character is entered, one of the program function or cursor control keys was probably pressed. In this case, there will be one or two additional characters that complete the escape sequence. The escape sequence for each of the PF and cursor control keys is mapped onto control characters. Seldom-used control characters were chosen to represent each key as follows: codes 0 thru 3 correspond to PF1 thru PF4, and codes 4 thru 7 correspond to cursor movement keys up/down/right/left. This representation of frequently-used escape sequences as single character control codes makes the APL code in the EDITSCREEN function simpler and more efficient.

The matrices XTBL and YTBL contain ASCII codes that form cursor-positioning escape sequences. Each of the 24 rows in XTBL corresponds to one row on the screen. Each of the 80 rows in YTBL corresponds to one column on the screen. The matrices are initialized by the SYSINIT function when the workspace is loaded. To support terminals other than the VT100, these tables would be modified to contain the escape sequences required by those terminals. In many APL systems, selecting one row from a matrix is a fast operation and so this table-driven approach to cursor control is very efficient.

EDITSCREEN: Editing a Matrix

The right argument SCR is a character matrix of any size that will fit on the display screen; this is usually smaller than 24 rows by 80 columns. The result R is a matrix of the same shape reflecting any changes made by the user.

Lines 4 and 5 initialize LBL, a vector of labels used to control the processing of control characters. Each element of LBL corresponds to a particular ASCII control character. The element of LBL at index 8, for example, is label L16. The control character 8 corresponds to backspace; the APL statements at label L16 perform the editing

```
    ∇ SYSINIT;N
[1]   ⎕IO←1
[2]   N← 49 50 51 52 53 54 55 56 57 48
[3]   XTBL← 24 4 ⍴0 ∨ XTBL[;1]←27 ◇ XTBL[;2]←91
[4]   XTBL[;3]←24↑1↓10/ 48 49 50 ◇ XTBL[;4]←24⍴N
[5]   YTBL← 80 4 ⍴0 ∨ YTBL[;1]←59
[6]   YTBL[;2]←80↑1↓10/47+⍳9
[7]   YTBL[;3]←80⍴N ∨ YTBL[;4]←72
[8]   ⎕ARBOUT 27 91 63 56 108 ⍝ AUTOREPEAT OFF
[9]   ⎕ARBOUT 27 91 63 52 108 ⍝ SMOOTH SCROLLING OFF
    ∇
```

```
    ∇ OUT TEXT
[1]   ⎕←TRANSLATE[⎕AV⍳TEXT]
    ∇
```

actions desired whenever a backspace is entered.

On lines 7 and 8 the following actions occur: TCFF clears the screen; WINDOWS establishes right and bottom margins for wrap-around; DETRANSLATE converts the input character matrix to a matrix of integers; OUT displays the character matrix on the terminal screen; and C is assigned the desired cursor address (0 0 corresponds to top row, left margin).

All changes to the contents of the matrix being edited are made on line L1. First INKEY displays the character CHAR at the starting cursor address, moves the cursor to location C, and awaits input from the user's keyboard via ⎕ARBIN. If the input is a control character, a branch to L2 occurs. Otherwise the matrix being edited is altered at location C to contain the input character. The desired cursor address is advanced one column with wrap-around occurring at the right-most column. Then line L1 is executed again; the first thing that happens is the transmission or echoing of the character just entered so that it appears at the current cursor location. The cursor is then moved to the new cursor address C and ⎕ARBIN is used to receive the next character. This process repeats until a control character is returned by INKEY.

Line L2 is executed whenever the user enters a control character, or whenever the user presses a key such as PF1 or "cursor left" that is mapped onto a control character by the INKEY function. Those control characters without a corresponding label in LBL cause a branch to BEEP. In this case, a BEL (7) bell character is transmitted to the terminal to indicate rejection of input, CHAR is assigned the empty vector so nothing will be echoed to the terminal, and line L1 is executed again.

Lines L10 thru L15 implement cursor control. Line L16 implements destructive backspace by inserting a 0 into the matrix being edited and assigning CHAR to be a blank (32). Line L19 is the exit line. The result R is converted from numeric to character and the function terminates. Line L19B implements the abort action in response to PF4 or control-C. Line L19A implements the new line function signalled by carriage return (control-M with ASCII code 13).

Limitations and Extensions

The arbitrary mapping of certain escape sequences onto seldom-used control characters has two disadvantages that should be mentioned although they have not proved serious at all so far. First, the usurption of control characters such as ENQ and ACK for other purposes precludes using them as intended by the ASCII standard. Second, the implementation of this approach in the INKEY function is specific to the VT100 terminal. The first objection did not prove to be a disadvantage in our application so we said "if it works, use it". The second objection can be overcome by extending INKEY to make it work with other terminals.

The limitation to matrices smaller than 24 by 80 could be relaxed by implementing a windowing mechanism where the window can be placed anywhere in an arbitrarily large matrix. For example, editing a letter-size matrix of 66 lines could be accomplished by adding 2 new lines to the EDITSCREEN function that would move the screen window toward the top or bottom of the document. Then the user would be able to view any 24 lines on the page by moving the screen window, in effect scrolling the letter thru the window.

Additional text-manipulating facilities such as an insert mode and a string-search capability could be added by using other control codes as the signals from INKEY for additional lines of EDITSCREEN to be executed. Thus this design can be easily extended to perform additional editing tasks on a VT100 and can be extended to other terminals as well. The slight inefficiency of using APL to process terminal I/O on a character-by-character basis is offset by the ease of modifying the editor to meet local needs.

```
    ∇ R←EDITSCREEN SCR;C;CHAR;LBL;WIND;ATTR;WINDOWS;
      TEXT;TITLE;⎕IO
[1]   ⍝*** FOR EDITING SCREENS
[2]   ⍝ MYERS  (19-JUNE-1983)  STSC
[3]   ⍝
[4]   ⎕IO←0 ⋄ LBL←32⍴BEEP
[5]   LBL[4 7 6 5 8 3 0 13 9]←L12,L13,L14,L15,L16,L1
      9B,L19,L19A,L10
[6]   ⍝---CONVERT TEXT TO NUMERICAL VALUES (=INKEY VA
      LUE) FOR EFFICIENCY
[7]   TCFF ⋄ WINDOWS←2↑⍴SCR ⋄ R←DETRANSLATE⍳SCR ⋄ OU
      T 22 80 ↑SCR ⋄ C← 0 0
[8]   ⍝
[9]   ⍝
[10]  CURSOR C ⋄ CHAR←⍳0
[11]  ⍝
[12]  L1:→(32>CHAR←CHAR INKEY C)⍴L2 ⋄ R[C[0];C[1]]←CH
      AR-32 ⋄ C←WINDOWS| 0 1 +C ⋄ →L1
[13]  L2:→LBL[CHAR] ⍝ CONTROL CHARACTERS
[14]  BEEP:⎕ARBOUT 7 ⋄ CHAR←⍳0 ⋄ →L1
[15]  ⍝
[16]  L10:C←WINDOWS| 0 8 +C ⋄ CHAR←⍳0 ⋄ →L1 ⍝ TAB CHA
      RACTER
[17]  L12:C←WINDOWS| ¯1 0 +C ⋄ CHAR←⍳0 ⋄ →L1 ⍝ CURSOR
       UP
[18]  L13:C←WINDOWS| 0 ¯1 +C ⋄ CHAR←⍳0 ⋄ →L1 ⍝ CURSOR
       LEFT
[19]  L14:C←WINDOWS| 0 1 +C ⋄ CHAR←⍳0 ⋄ →L1 ⍝ CURSOR
       RIGHT
[20]  L15:C←WINDOWS| 1 0 +C ⋄ CHAR←⍳0 ⋄ →L1 ⍝ CURSOR
       DOWN
[21]  L16:C←WINDOWS| 0 ¯1 +C ⋄ R[C[0];C[1]]←0 ⋄ CURSO
      R C ⋄ CHAR←32 ⋄ →L1 ⍝ BS
[22]  ⍝
[23]  L19:TCFF ⋄ R←DETRANSLATE[R] ⋄ →0 ⍝ END KEY PRES
      SED  PF1
[24]  L19B:R←,'' ⋄ TCFF ⋄ →0 ⍝ ESC  PF4
[25]  L19A:C←(WINDOWS[1]|1+1↑C),0 ⋄ CHAR←⍳0 ⋄ →L1 ⍝ C
      ARRIAGE RETURN
    ∇
```

OTHER THINGS TO DO IN VEGAS

Douglas Bohrer
Editor, The Special Character Set

The APL SIG will have a shared hospitality suite in Las Vegas. The suite will come complete with VT102 with APL characters on it. Thanks are due to Jeanne Brattlof, DEC's marketing guru for "The APL Video Terminal." Upgrade kits are available for regular VT102s.

The suite will be shared with the MUMPS SIG. If you haven't had it before, get a vaccination.

The RT-11 SIG is sponsoring a "DECUS Library Products Panel for RT-11." Just like DEC's product panel sessions, this one will talk about all of the DECUS "Layered Products" for RT-11 including APL-11. It's scheduled for Monday 5:30-7:00PM.

Functions and Idioms

MULTIPLE LINEAR REGRESSION IN APL-11

Douglas Bohrer
Bohrer & Company
903 Ridge Road Suite 3
Wilmette, IL 60091

This article describes the steps used to develop programs in APL-11 which together form a general purpose regression package useful for any type of multiple linear regression. The article also covers how to use the programs in various situations. Some of the functions described are in standard APL and could be used on any APL system, although they might need some modification to run efficiently.

The Problem

This problem began like most problems. One of my colleagues came into my office at the bank and said that the manure had hit the fan. It turned out that a vice president in Compensation was murmering darkly about lots of regressions he wanted to do. Since I had the only master's degree in statistics in the unit, it was my job to find out what he wanted and see if we could do it. He wanted to do some small exploritory linear regressions first, and work up to the entire Salary Administration file later. The regressions were to be all possible combinations of seven to ten independent variables in both linear and log linear models. No sweat. Except that we had no quick way to get to a statistical package and also conform to the confidentiality policy for the data. I had to write a package to do the work on the totally secure PDP-11 in less than a month.

Regression with Data in the Workspace

I had done some regressions before for this vice president, but the data always fit in the 29KB workspace available to me in APL-11 under TSX-PLUS. The functions I used for this work were REGR and REGRPRT. The formulas in REGR were taken directly from standard texts on regression, translated directly from the matrix notation in the book to APL. The only difference between the REGR I used before and the one listed below is that I used the domino function, called DMD in the APL-11 release file INVERT.APL to calculate BETA instead of the current lines 2 and 3.

There were symbol table problems with using DMD. In the previous work these didn't make much difference because there were only a couple of regressions to do. I worked around the problems mannually. This time I knew I might have to do over 100 regressions. This meant I had to fix DMD or write a new function entirely. I decided to write a Gaussian Elimination function to invert square matrices. The algorithm is simpler than DMD, less numerically stable and much less general. It seemed to be an easier way to go than fixing DMD. The function, INVT is listed below. It took far less symbol table space than DMD, ran much faster and seemed to be just as accurate for what I needed to do.

Since I was going to have a lot of runs, I needed to send the output to a file. I modified REGPRT to send all of the output to channel 12 and allow for a comment line on the output to tell me what the variables in each regression were. The resulting function is called REGOUT. The function used to send the output to a file, ROUT, is described in The Special Character Set No. 1, February 27, 1981, page 8. The listing of ROUT below is the current version. CCNO is a 2 element vector of the control N and control O characters (shift in and shift out). There is no thorn formatting function in APL-11v1 so it is simulated with my function THORN described in "A Thorn for APL-11 V1", The Special Character Set No. 1, February 27, 1981, page 7.

All of the above was sufficient to do small regressions with data in the workspace. There remained the problem of using several thousand data points from the Salary Administration file.

BIG Regressions

To do a really large regression I had to modify REGR to use a data file instead of workspace variables. The resulting function, REGF, is called with the channel number for the file, CC, and the collumn numbers for the dependent and independent variables, XX. REGF follows the methods described in "BIG Calculations in APL-11", The Special Character Set No. 5, March 15, 1983, pages 4-6.

The APL initialization for the passes through the data file is done in lines 1-5,10. YV, the collumn for the independent variable, must be a scalar so that when it is used to index data from the file the result is a vector. Both YV and XV, the dependent variable collumns, must be global variables to avoid having to pass them as function arguments. The variables set in lines 4, 5 and 10

```
     ∇ YY REGR XX
[1]   ZZ←1,MTX XX
[2]   →(1=ρ,INV←INVT (⍉ZZ)+.×ZZ)/0
[3]   BETA←(INV+.×⍉ZZ)+.×YY
[4]   RST←((SSE←+/(YY-+/((ρZZ)ρBETA)×ZZ)*2)÷FRE←-/RO←ρZZ)*0.5
[5]   TST←BETA÷BST←(1 1 ⍉VAR←(RST*2)×INV)*0.5
[6]   RSQ←(SSR←SST-SSE)÷SST←+/(YY-(+/YY)÷ρYY)*2
[7]   ADJ←RSQ-((RO[2]-1)÷FRE)×(1-RSQ)
[8]   FST←(SSR÷RO[2]-1)÷(SSE÷-/FRE)
     ∇
```

```
     ∇ OUT←MTX IN
[1]   →(1<ρρOUT←IN)/0
[2]   OUT←((ρ,IN),1)ρIN
     ∇
```

```
     ∇ REGPRT ;L
[1]   'REGRESSION ANALYSIS'
[2]   →(0≠ρ⍞←((⍳0)ρ1=ρ,INV)/'SINGULAR MATRIX: NO REGRESSION POSSIBLE')/0
[3]   'ADJUSTED R-SQ.  =',,8 6 THORN ADJ
[4]   'DEGREES OF FREEDOM =',,9 THORN FRE
[5]   ' '
[6]   'R-SQUARE =',,6 4 THORN RSQ
[7]   'F = ',,7 3 THORN FST
[8]   'SUM OF SQUARED ERROR =',,⍕ SSE
[9]   ' '
[10]  '                        BETA       S.E BETA        T-STAT'
[11]  ' '
[12]  L←(((ρBETA),12)ρ12↑'VARIABLE NO.'),2 0 THORN((ρBETA),1)ρ¯1+⍳ρBETA
[13]  L[1;]←14↑'CONSTANT'
[14]  L,14 5 THORN BETA,BST,[1.5]TST
     ∇

     ∇ R←INVT M ;CC;I;J
[1]   J←1
[2]   R←M,(⍳1↑ρM)∘.=⍳1↑ρM
[3]   B1: →(5E¯7  >|CC[I←(|CC)⍳⌈/|CC←(((J-1),0)↓R)[;1]])/B2
[4]   CC←R[J+I-1;]
[5]   R[J+I-1;]←R[J;]
[6]   R[J;]←CC
[7]   CC←R[;1]
[8]   R[J;]←R[J;]×÷CC[J]
[9]   CC[J]←0
[10]  R←(0 1↓R)-CC∘.×1↓R[J;]
[11]  →((ρM)≥J←J+1)/B1
[12] B2: →(∧/(ρM)=ρR)/0
[13]  'SINGULAR MATRIX'
[14]  R←1
     ∇

     ∇ CM ROUT VF ;I
[1]   I←1
[2]   →(2=ρρCM)/B1
[3]   CM←(2↑(⍳2>ρρCM),(ρCM),1)ρCM
[4]  B1: FP←0/⍷(⍕VF),'⌹CM[I;]'
[5]   →((1↑ρCM)≥I←I+1)/B1
     ∇

     ∇ REGOUT ;L
[1]   CCNO[1] ROUT 12
[2]   'ENTER COMMENT LINE'
[3]   (   'REGRESSION ANALYSIS') ROUT 12
[4]   ⍞ ROUT 12
[5]   →(1≠ρ,INV)/B1
[6]   'SINGULAR MATRIX: NO REGRESSION POSSIBLE' ROUT 12
[7]  B1: →(1=ρ,INV)/B2
[8]   (   ' ') ROUT 12
[9]   (   'ADJUSTED R-SQ.  =',,8 6 THORN ADJ) ROUT 12
[10]  (   'DEGREES OF FREEDOM =',,9 THORN FRE) ROUT 12
[11]  (   ' ') ROUT 12
[12]  (   'R-SQUARE =',,6 4 THORN RSQ) ROUT 12
[13]  (   'F = ',,7 3 THORN FST) ROUT 12
[14]  ('SUM OF SQUARED ERROR =',,⍕ SSE) ROUT 12
[15]  (   ' ') ROUT 12
[16]  (   '                        BETA       S.E BETA        T-STAT') ROUT 12
[17]  (  ' ') ROUT 12
[18]  L←(((ρBETA),12)ρ12↑'VARIABLE NO.'),2 0 THORN((ρBETA),1)ρ¯1+⍳ρBETA
[19]  L[1;]←14↑'CONSTANT'
[20]  (  L,14 5 THORN BETA,BST,[1.5]TST) ROUT 12
[21] B2: CCNO[2] ROUT 12
[22]  (3 1ρ' ')ROUT 12
     ∇

      ∇THORN[⍞]∇
     ∇ R←F THORN M ;I;RO;T
[1]   F←F,(1=ρ,F)/0
[2]   RO←(⍳1=ρρM),(⍳0=ρ,ρM),ρM←(10*-F[2])×⌈¯0.5 +M×10*F[2]
[3]   M←((×/RO),1)ρM
[4]   M←(0 ¯2+ρM)↑M←((×/RO),(ρM)↑×/RO)ρM←(⍕M),'  '
[5]   T←(×/RO)ρF[2]-F[2]=0
[6]   I←1
[7]   B1: T[I]←(~'E'∈M[I;])×T[I]-(¯1↑ρM)-(M[I;]⍳'.')⌊1+(1↓M[I;])⍳' '
[8]   →((ρT)≥I←I+1)/B1
[9]   R←(RO×1,1↑F)ρM←((1↑ρM),-1↑F)↑M←TΦM←(((1↑ρM),F[1]-1)ρ' '),M
     ∇
```

(Function REGF is on the next page)

are zeros with the appropriate shape so that results can be added up as the file is read.

The data on channel CC is all double precision numbers. The data is considered to be a big matrix, with each fixed-length record a row of the matrix. The function RES reads the data into global variable "A" several rows at a time so that at each itteration the data fits in the workspace. After reading some data at each itteration, RES executes its left argument which is a character string. Each of the left arguments in REGF (lines 6, 9 and 11) is a function which references the file data in variable "A". RES is explained more fully in "File Handling Helps in a Small Workspace" in The Special Character Set No.2, September 1, 1981, pages 18-20.

Three Passes Through the Data

PASS1 calculates the sum of the independent variable values and the inner product of the data matrix. A nice trick in line 2 is that the inner product of the whole data matrix on the file turns out to be the sum of the inner products of each piece of the data. As with all functions executed by RES, PASS1 adds the results from each piece of the data to accumulation variables. In this case the accumulation variables are YAV, the sum of the independent variable values, and INV, the inner product of the data matrix.

After the first pass, the average of the independent variable values and the inverse of the inner product of the data matrix are calculated in lines 7 and 8 of REGF. If the matrix is singular, REGF exits at line 7.

PASS2 calculates the regression coefficients, variable BETA. It also calculates the sum of the squared differences of the independent variable and its mean. Again, the sum of the inner products at each itteration is the inner product of the whole. Notice that PASS2 uses YAV and INV which were calculated from the results of PASS1.

PASS3 calculates the sum of the squared error. It uses BETA, which was calculated in PASS2.

There is a quirk of APL-11 which needs explaining in PASS1 and PASS3. The variable ZZ is a global variable which is erased as the last step in both. APL-11v1 does not reliably handle local variables with the same names as global variables.

```
      ∇ PASS1
 [1]   YAV←YAV++/A[;YV]
 [2]   INV←INV+(⍉ZZ)+.×ZZ←1,A[;XV]
 [3]   ∈')ERASE ZZ'
      ∇

     ∇ PASS2
[1]   SST←SST++/(A[;YV]-YAV)*2
[2]   BETA←BETA+(INV+.×⍉1,A[;XV])+.×A[;YV]
     ∇
```

Since ZZ is used in REGR as a global, the potential problem exists. Using another name would take up scarce symbol table space. Leaving it unerased would fragment the 29KB workspace, and there is no garbage collection in APL-11.

Setting Up the Data

At this point I hoped I had done enough coding to do the regressions themselves. But the vice president had thrown me a curve. He wanted log-linear as well as linear regressions. Not only that, but he wanted indicator variables for review ratings which ranged from 1 to 7. (The indicator variable for rating 7 is 1 if the person got a 7 and 0 otherwise.) Both of these calculations are easy to do if the data fits into the workspace. For log-linear, you simply do a linear regression with the log of the independent variable. Indicator variables are quite easily done with a jot dot equals outer product. But how do you do these kinds of things to a big file?

My solution was to write a general purpose utility to read input file(s), do some APL calculations like RES does and write the results to an output file. The function FRES is a modification of RES which reads through the input file(s) a piece at a time and then executes its character string left argument against each piece of the data. After the execution of the string, whatever is in variable FR is written to the output channel. Using FRES, I set up the file for input to REGF with a function very similar to what I would have used to set up workspace variables.

```
      ∇ CC REGF XX
 [1]   CC←,CC
 [2]   YV←(⍳0)⍴1↑XX
 [3]   XV←1↓XX
 [4]   BETA←(⍴XX)⍴0
 [5]   INV←(2⍴1+⍴XV)⍴YAV←SST←0
 [6]   'PASS1'RES CC
 [7]   →(1=⍴,INV←INVT INV)/0
 [8]   YAV←YAV÷STR[CC[1];2]
 [9]   'PASS2' RES CC
 [10]  SSE←0
 [11]  'PASS3' RES CC
 [12]  RST←(SSE÷FRE←-/RO←STR[CC[1];2 3])*0.5
 [13]  TST←BETA÷BST←(1 1 ⍉VAR←(RST*2)×INV)*0.5
 [14]  RSQ←(SSR←SST-SSE)÷SST
 [15]  ADJ←RSQ-((RO[2]-1)÷FRE)×(1-RSQ)
 [16]  FST←(SSR÷RO[2]-1)÷(SSE÷-/FRE)
      ∇
```

```
      ∇ PASS3
 [1]   SSE←SSE++/(A[;YV]-+/((⍴ZZ)⍴BETA)×ZZ←1,A[;XV])*2
 [2]   ∈')ERASE ZZ'
      ∇
```

```
     ∇ CM FRES VF
[1]   PR←1
[2]   RF←VF[2]
[3]   TSTR←2 4⍴VF[1],0 1 25,STR[RF;]
[4]   VF←2↓VF
[5]   ROR←⍳0
[6]   CM←(2↑(⍳1=,⍴⍴CM),(⍴CM),1 1)⍴CM
[7]   PL←1,STR[VF[1];2]
[8]   →(3≥⍴VF)/B20
[9]   PL←¯2↑VF
[10]  VF←,(0≠VF)/VF←¯2↓VF
[11] B20: P←PL[1]
[12]  BL←(1+PL[2]-PL[1])⌊⌊SIZ÷+/STR[VF;3]×8-6×STR[VF;1]=3
[13] B30: →B30+4-⍴VF
[14]  →(0=1↑⍴C←GET VF[3],P,BL)/0
[15]  →(0=1↑⍴B←GET VF[2],P,BL)/0
[16]  →(0=1↑⍴A←GET VF[1],P,BL)/0
[17]  LCR←1
[18] B40: FP←0/∈CM[LCR;]
[19]  →((LCR←LCR+1)≤1↑⍴CM)/B40
[20]  ∈')ERASE A B C'
[21]  →(0=×/1,⍴FR)/B70
[22]  ROR←⍴FR
[23]  STR[RF;]←TSTR[1;]
[24]  FR PUT RF,PR
[25]  STR[RF;]←TSTR[2;]
[26]  PR←PR+×/1,ROR
[27]  ∈')ERASE FR'
[28]  →(~RF∈VF)/B70
[29]  →(~(PR×BYT[TSTR[1;1]])>PF×STR[RF;3]×BYT[STR[RF;1]])/B70
[30]  'FRES WARNING: POSSIBLE WRITE BEFORE READ'
[31] B70: →(PL[2]>¯1+BL+P←P+BL)/B30
[32]  →(0<BL←1+PL[2]-P)/B30
[33]  TSTR[1;2 3]←2↑ROR,(1=⍴ROR),0=⍴ROR
[34]  TSTR[1;2]←(PR-1)↑TSTR[1;3]
[35]  STR[RF;]←TSTR[1;]
[36]  ∈')ERASE LCR PL FP BL P PR RF TSTR ROR'
     ∇
```

The Results

I was able to work through the entire process above and complete 180 regressions in less than two weeks. The actual design, coding and testing took about half of that time. I think this is a very good demonstration of the power of APL, even a very limited implementation like APL-11. Run time for each regression was about 5 to 10 minutes on a PDP-11/23 depending on the number of variables and data points. There were an average of 4 variables and 2000 data points in each regression.

The vice president was very pleased and wrote a letter to my boss saying so.

Details

All of the functions used in this article are available on the Spring 1983 SIG Symposium tapes for RT-11 and RSTS/E. Copies of APL-11 version 1 are also available on these tapes. A catalogue of what Larry LeBlanc and I put on these tapes is printed in this issue. Symposium tapes are available through your Local User Group. They are usually copied free if you supply the media. The tapes are also available from the DECUS Library for about $150. Check with them for details.

I have also submitted the entire collection for RT-11 to the DECUS Library. I will probably know the order number in time for the next issue.


```
     ∇ R←GET VCS
[1]   FP←STR[VCS[1];4]+BYT[STR[VCS[1];1]]×STR[VCS[1];3]×¯1+VCS[2]
[2]   FP←∈(⍑VCS[1]),'⊞FP'
[3]   R←∈(⍑VCS[1]),'⊞[STR[VCS[1];1]]VCS[3]×STR[VCS[1];3]'
[4]   →(0=1↑⍴R)/0
[5]   R←(VCS[3],(STR[VCS[1];3]≠1)/STR[VCS[1];3])⍴R
     ∇
     ∇ R PUT VCS
[1]   FP←STR[VCS[1];4]+BYT[STR[VCS[1];1]]×STR[VCS[1];3]×¯1+VCS[2]
[2]   FP←∈(⍑VCS[1]),'⊞FP'
[3]   FP←0/∈(⍑VCS[1]),'⊞[STR[VCS[1];1]]R'
     ∇
     ∇ CM RES VF
[1]   VF←,VF
[2]   CM←(2↑(⍳1=,⍴⍴CM),(⍴CM),1 1)⍴CM
[3]   PL←1,STR[VF[1];2]
[4]   →(3≥⍴VF)/B20
[5]   PL←¯2↑VF
[6]   VF←,(0≠VF)/VF←¯2↓VF
[7]  B20: P←PL[1]
[8]   BL←(1+PL[2]-PL[1])⌊⌊SIZ÷+/STR[VF;3]×8-6×STR[VF;1]=3
[9]  B30: →B30+4-⍴VF
[10]  →(0=1↑⍴C←GET VF[3],P,BL)/0
[11]  →(0=1↑⍴B←GET VF[2],P,BL)/0
[12]  →(0=1↑⍴A←GET VF[1],P,BL)/0
[13]  LCR←1
[14] B40: FP←0/∈CM[LCR;]
[15]  →((LCR←LCR+1)≤1↑⍴CM)/B40
[16]  ∈')ERASE A B C'
[17] B70: →(PL[2]>¯1+BL+P←P+BL)/B30
[18]  →(0<BL←1+PL[2]-P)/B30
[19]  ∈')ERASE LCR PL FP BL P'
     ∇
```

State Indicators

APL ON RT-11 AND RSTS/E SIG TAPES

Doug Bohrer and Larry LeBlanc
APL SIG

Larry and Doug have each put together a tape submitted to their respective operating system SIGs for inclusion on the Spring 1983 Symposium tapes. The tapes submitted include APL-11 version 1 and some useful utilities. Version 1 is the most reliable version available for RT-11 and the only version available for RSTS/E.

US SIG Symposium tapes are available through your Local User Group. They are usually copied free if you supply the media. The tapes are also available from the DECUS Library for about $150. Check with them for details.

Doug has also submitted the entire collection for RT-11 to the DECUS Library. He will probably know the order number in time for the next issue. What follows is a listing of the APLUTL.DOC and FORUTL.DOC files found on both the RT-11 and RSTS/E tapes. The directory of the RSTS/E submission comes next. Following that are listings of the READ*.* and *.HLP files from the RSTS/E tape.

```
C
                    APL UTILITY DOCUMENTATION



      For DIGITAL software documentation order the "APL Programmer's
Reference Manuel AA-5076A-TC APL-11/RSTS/E V1".
THAT'S RIGHT FOLKS, WE MEAN RSTS/E.


**************************************************************************

CHOOSING THE RIGHT FILE FOR RT-11:
Hardware Considerations:
      1. no EXTENDED INSTRUCTION SET use APL00 and APL01.

      2. have FLOATING-POINT INSTRUCTION SET use APL04.

      3. have no FLOATING POINT PROCESSOR,but do have EIS use
         APL02 and APL03.

      4. have FLOATING POINT PROCESSOR AND EIS use APL06 and APL07.


Precision

      single-APL00,APL02,APL04,APL06

      double-APL01,APL03,APL07

Are you scratching your head still about which one,well just try the higher
numbered SAV file first because it will have the most features.
```

****

The APL utilities have been divided into three sections. Section one has those involving calculations and section two is for printing. Section three has statistical regression programs.

To install these utilities for daily use you will have to write them out to separate files. It is convenient to group functions commonly used together into single workspaces. At the bank, we have the following groups:
GET,INSTR,PUT,SHUT,TO,BYT,SIZ,STR
CMAT,RJUST
ROUT,THORN,CCNO,FF
The other functions each are in their own separate files so we can)READ them in only if we need them. We typically)READ them into each program that uses them, so we only have to maintain one copy.

These functions were developed under RT-11 (TSX-PLUS). Some have not been tested under RSTS/E. The ones that have been tested under RSTS/E worked.

==========
==========
==========

THE FOLLOWING SECTION DOCUMENTS UTLCAL.APL

These are the functions in 'UTLCAL' file which stands for utility calculations.

CNTCOL was written by Don Scheer and Roy A. Sykes, Jr.
(STSC, INC., Woodland Hills, CA)

JUL and YEARS were written by Russ Seward.
(First National Bank of Chicago)

All other functions were written by Doug Bohrer.
(First National Bank of Chicago)

CNTCOL,GET, PUT AND RES are described in back issues of the APL SIG Newsletter,"The Special Character Set" which are (or soon will be)) available from DECUS.

****
****

function name: ADDCOL

purpose: This function totals the values for each pair of numbers in the first two columns of a 3-column matrix. (The third column has the values to be totaled.)

to call: The input has three parts. A global variable called "COL" contains the possible numbers for the second column of the input matrix. The possible characters in the first column of that matrix are in the left argument. The right argument is a three column matrix that contains the pairs of numbers that will be tabulated.
The output is a tabulation matrix. The matrix has as many rows as the numbers of values in the left argment plus one more, and as many columns as the number of values in "COL" plus one more. The "one more" is for values found in the pairs that are not listed in the left argument and/or "COL" . In other words, the additional row and column are for unknowns. An ADDCOL with the 3rd column of the matrix all 1's will produce a result the same as a CNTCOL with the first 2 columns.

****

****

function name: CNTCOL

purpose: This function tabulates the number of times a certain pair of characters or numbers occur in the row of a two-column matrix. As a bonus, the function also keeps a count of the bad data.

to call: The input has three parts. A global variable called "COL" contains the possible characters or numbers for the second column of the input matrix. The possible characters in the first column of that matrix are in the left argument. The right argument is a two-column matrix that contains the pairs of characters of numbers that will be tabulated.
The output is a tabulation matrix. The matrix has as many rows as the number of values in the left argument plus one more, and as many columns as the number of values in "COL" plus one more. The "one more" is for values found in the pairs that are not listed in the left argument and/or "COL".
In other words, the additional row and column are for unknowns.

****
****

function name:FRES

purpose: This program manages the loop which reads through a file or files separately from the calculations to be done at each iteration. At each iteration a calculation result is written to an output file.

to call: The call is in the form:
'character string' FRES data type,output channel, input channels
One to three input channels are allowed. The character string is usually the name of the APL function that does the calculations, but it may be any executable statement. The output channel may be one of the input channels as long as the output data is not longer than the data input from the channel. The input data is read into the global variables A, B and C from the first, second and third input channels. The calculations then reference A, B and C. Variables B and C are not set if there is no channel number for them. Global variable FR is written to the output channel and MUST be set in the calculations. If FR is null for any iteration, nothing is written. Variables SIZ, STR, BYT and functions GET and PUT are used. Function RES uses the same variables as FRES so they can't be called from each other.

****
****

function name:GET

purpose: This function gets a piece of the file. "GET" uses the structure of the file (variable "STR" has the structral information) to structure the result. The number of rows (or vector elements) is that specified in the function call.

to call: The right argument is a vector containing the channel number start row and the number of rows of the file to be gotten.
The result is that chunk of the file.
The variable "STR" is a matrix with each row for each channel(1-5).
The columns of "STR" have this info:col■1 is data type,col■2 is number of records per fileds,col■3 is number of elements per record, col■4 is the start byte for data.

****

****

function name: INSTR

purpose: This function strips off the structural information of an APL structural file.
The structural info is stored in the matrix "STR" in the row corresponding to the channel number.

to call: The right argument is the channel or channels of the files that structural info is needed. APL structured files have the type,number of records and number of elements per record stored in the first 3 double precision floating point numbers on the file. Data begins in byte 25.

****
****

function name: EPSI

purpose: this function searches for any occurence of the rows of one matrix in the rows of another.

to call: The call is similar to the call for the standard APL vector membership: MATRIX1 EPSI MATRIX2 will give a logical vector with a one in the positions of the rows of MATRIX1 which occur as rows in MATRIX2.

****
****

function name: IOTA

purpose: This function searches for the first occurence of the rows of one matrix in the rows of another.

to call: The call is similar to the call for the standard APL vector iota: MATRIX1 IOTA MATRIX2 will give the indexes for the rows of MATRIX1 which are the first occurences of the rows of MATRIX2.

****
****

function name: JUL

purpose: This function calculates the julian number for any date in the form month,day,and year. For bad dates the result is 0.

to call: The right argument is matrix with three-columns having this date information month,day,and year in numeric form only.
The result is a vector of the julian numbers for the dates.
Function must have variables MONT,DAYS.

****
****

function name: PUT

purpose: This functon writes data out to a file in chunks.

to call: The right argument is a vector with this information (channel number and starting row). The left argument is the data to be written out.
This function uses variables "STR" and "BYT".

****

Tapes

function name:RES

purpose: This function manages the loop which reads through a file or files separately from the calculations to be done at each iteration.

to call: The call is in this form:
'character string' RES one to three channel numbers
The character string is usually the name of a function which does the calculations and accumulates results, but it can be any executable APL statement. The data is read into global variables A, B and C from the first, second and third channel numbers respectively. The calculations then reference A, B and C. Variables B and C are not set if there is no channel number for them. Variables STR,SIZ,BYT and function GET are used. Function RES uses the same variables as function FRES so they can't be called from each other.

****

function name: SHUT

purpose: This function closes a channel with an STR.

to call: The right argument is a vector of the channels to be closed.
The variable STR is used.
The contents of the first 3 columns of STR are written to the file if and only if the fourth column is 25.

****

function name: TO

purpose: This function produces a vector of consecutive positive integers starting with the left argument and ending with the right argument.

to call: 3 TO 6 is 3 4 5 6.

****

function name: YEARS

purpose: This function calculates the time between one fixed date and several other dates. Time is measured several ways:
total years (rounded down)
days past last anniversary date
years and fraction of years

to call: The right argument is the fixed (numeric) date: MM DD YY.
The left argument is a matrix of the dates for comparison where every row is a numeric MM DD YY. The result is a 3 column numeric matrix. Each row corresponds to the rows of the left argument. The 3 columns of the result are as described in the purpose above.

****

==========
==========
==========

THE FOLLOWING SECTION DOCUMENTS UTLPRT.APL
These functions and variables are all related to print output.
All of these functions were written by Doug Bohrer.
BARS RJUST ROUT and THORN are described in back issues of the APL SIG Newsletter "The Special Character Set" which are (or soon will be) available from DECUS.

****

function: BARS

purpose: Generates a bar graph.

to call: The left argument is a character string. Each character in the string is used for one bar of the graph in order of appearance repeatedly until all the bars are finished. The right argument is converted to a vector of counts for the bars. The 100 in lines two, five and eight is the width of the graph and may be changed.

****

function name: C2B

purpose: This function trims embedded blank strings to two blanks.

to call: C2B 'A B C D E' returns 'A B C D E'

****

function: CMAT

purpose: Creates a character matrix of the values for each row that you enter, trimming off all trailing blank columns.

to call: The right argument is the number of rows you want to enter.

****

function: MBAR

purpose: Generates fancy bar graphs.

to call: The left argument is a character string of characters to use in the graph. The right argument is a matrix of counts to be displayed. A bar is generated for each row of the matrix. Each column is the number of characters of the corresponding position in the left argument to be generated. The number of characters in the string should be equal to the number of columns in the matrix.

****

function: RJUST

purpose: To create right justified column headings.

to call: The left argument is the field widths to be used. The right argument is a character string of the headings separated by the branch arrow character.

****

function: ROUT

purpose: Moves character data to a print image file.

to call: The left argument is the character scalar, vector or matrix to be moved to the print image file. The right argument is the channel number of the file.

****

Tapes **********

function: THORN

purpose: Changes numeric data to character data

to call: The left argument is a 2 element vector containing the size of the field and the number of digits to the right of the decimal point. The right argument is numeric data. This function will round the data off.
The call has this form:
(size of field, decimal places) THORN data

****
variable: CCNO

purpose: Contains a control O and a control N for changing the character set. CCNO[1] switches to the APL character set.
CCNO[2] switches to the standard character set.

****
variable: FF

purpose: FF has the form feed character.

==========
==========
==========
THE FOLLOWING SECTION DOCUMENTS REGRES.APL
These functions are all related to statistical regression. They were all written by Doug Bohrer. Some of the functions use functions found in the other two sections above. Doug may get around to documenting these in future issues of "The Special Character Set" so please subscribe to it.
(Roger Matus may recognize REGR and REGPRT as somewhat similar to programs he wrote while working at The First National Bank of Chicago.)
IT IS STRONGLY RECOMMENDED THAT THESE FUNCTIONS BE USED ONLY WITH DOUBLE PRECISION. In single precision, REGF may loose a lot to round-off error and the THORN calls in REGOUT and REGPRT may not work due to E format representations.

****

function name: INVT

purpose: This function inverts a square matrix.

to call: The call is identical to that for APL monadic domino:
INVT square matrix
The inverse is returned unless the matrix is singular. If the matrix is singular, an error message is printed and a scalar 1 is returned. This function is faster than the function MMD which DEC supplied with APL-11, and it takes up less symbol table space and workspace. It also tends to be slightly more accurate.

****

function name: MTX

purpose: This function converts a vector or scalar into a single column matrix but returns matrices unchanged.

to call: The call is in the form:
MTX scalar or vector or matrix
The result is a matrix.

****

function name: PASS1

purpose: Calculates the sum of the values of the independent variable and the inner product of the data matrix.

to call: Called from RES inside REGF. Of no use by itself.

****

function name: PASS2

purpose: Calculates the sum of the squared differences of the independent varaible and its mean. Also calculates the regression coefficients (BETA).

to call: Called from RES inside REGF. Of no use by itself.

****

function name: PASS3

purpose: Calculates the sum of the squared error.

to call: Called from RES inside REGF. Of no use by itself.

****

function name: REGF

purpose: Calculates a statistical regression (simple or multiple) from data on a numeric binary file.

to call: The call takes the following form:
channel number REGF columns of independent, dependent varaibles
Uses functions INVT,PASS1,PASS2,PASS3 in this file and functions RES and GET and variables BYT,SIZ,STR from file UTLCAL.APL. Sets up several variables for use by REGPRT or REGOUT.

****

function name: REGOUT

purpose: This function formats the results of a regression and writes them to an output file.

to call: The call is in the form:
REGOUT output channel number
REGOUT uses variables made by REGR or REGF. REGOUT also uses functions ROUT and THORN from file UTLPRT.APL.

****

function name: REGPRT

purpose: This function formats the results of a regression and prints them on the terminal.

to call: The function is niladic and returns nothing. It uses variables set by REGR or REGF and function THORN from file UTLPRT.APL.

****

function name: REGR

purpose: Calculates a statistical regression (simple or multiple) from data in the workspace.

to call: The call is of the form:
independent variable REGR dependent variable(s)
This function uses function MTX and INVT.

****
****
****
END OF APLUTL.DOC

FORTRAN AND C UTILITY DOCUMENTATION

****

Neither First Chicago Corporation, the First National Bank of Chicago, nor any of its employees makes any warranty, either expressed or implied, assumes any legal liability, nor any responsibility for the accuracy, completeness or usefulness of any information, product or process disclosed. Neither does First Chicago Corporation, the First National Bank of Chicago, nor any of its employees represent that use of this material would not infringe privately owned rights.

****

FORTRAN AND C UTILITY DOCUMENTATION

This document briefly discribes the purpose of the following utilities:

FORTRAN UTILITIES ..fixlen,fsalen,charex,taper,spdump,mtdump, cread,match,matchb

C UTILITIES ..fixlen,fsalen,mergef,high density backup system

Throughout the remainder of this document, the files named '*.FOR' are FORTRAN programs, '*.C' are C programs, '*.SAV' are executable C or FORTRAN programs, and '*.HLP' are the HELP documentation for the named program.

All of these utilities were developed and are currently being used on a DEC PDP 11/23 using:

RT-11 V4.0
TSX-PLUS V2.2
FORTRAN RT-11 V2.5
DECUS C COMPILER

RSTS/E USERS: Only the following programs are included for you:
fixlen,fsalen,charex,match,matchb,mergef
The other stuff doesn't seem to be useful for you. If you want it, you can get it on the RT-11 SIG tape. The *.sav files have not been tested under the RSTS/E RT-11 emulator.

****

FILES : CHAREX.FOR PROGRAM TO CONVERT A FSA FILE TO A TY4 FILE
CHAREX.SAV
CHAREX.HLP

PURPOSE: This program reads files of character data written in FORTRAN ASCII Sequential format and converts the data into single precision floating point numbers with APL structural data in the first 24 bytes on the output file.

****

FILES: FIXLEN.FOR PROGRAM TO CONVERT A FSA FILE TO A XSA FILE
FIXLEN.SAV
FIXLEN.C
FIXLEN.HLP

PURPOSE: FIXLEN changes a fortran sequntial ASCII(fsa) file into a fixed length ASCII (XSA) file.

The C and Fortran versions of this program have the same input and output requirements, therefore, the help document is used for both versions. The compiled file (FIXLEN.SAV) was compiled from FIXLEN.C.

****

FILES: FSALEN.FOR PROGRAM FOR FINDING THE LENGTH OF A FSA FILE
FSALEN.C
FSALEN.SAV
FSALEN.HLP

PURPOSE: This program counts the number of records in a Fortran Sequential ASCII (FSA) file and stores the result in an APL read file. Each record is examined a character at a time until the EOR is found. The number of records read is then written to the output file and all files are closed.

From APL, command)READ Output_file_name will read in the file. Variable LEN contains the length of the file.

This program is being presented in both FORTRAN and C. The HLP Documentation is good for either version. And the compiled file (FSALEN.SAV) is the C version.

****

FILES: MATCH.FOR PROGRAM FOR FILE COMPARISON
MATCH.SAV
MATCH.HLP

PURPOSE: This program compares the records of one presorted fortran sequential ASCII file to the records of another presorted fortran sequential ASCII file by checking their keys a character at a time. if both keys "match" (i.e., are the same) then both records are combined together and written to the output file for matched records. if the keys do not match then the record with the lower key value is written to the error output file along with an error message generated by the user at beginning of the program execution.
Output file includes only matched records.

****

FILES: MATCHB.FOR PROGRAM FOR FILE COMPARISON
MATCHB.SAV
MATCHB.HLP

PURPOSE: This program compares the records of one presorted fortran sequential ASCII file to the records of another presorted fortran sequential ASCII file by checking their keys a character at a time. if both keys "match" (i.e., are the same) then both records are combined together and written to the output file for matched records. if the keys do not match then the record with the lower key value is written to the error output file along with an error message generated by the user at beginning of the program execution. If this record was from file 1, then a record of blank characters is added to its end and this larger record is then written to the output file for matched records.

If the record was from file 2, however, then a record of blank characters is added to its beginning and this larger record is written to the output file for matched records.
Output file includes matched and unmatched records together.

****

FILES: MERGEF.C MERGING TWO PREVIOUSLY SORTED FSA FILES TOGETHER
MERGEF.SAV
MERGEF.HLP

PURPOSE: This program will merge parts or all of two sorted FSA files into one file. The two input files are matched and merged together based on the value of their respective sort key. These sort keys must be the same variable. The HLP documentation discribes in detail the input criteria required to run this program.

****

FILES: DUMPMT.FOR DUMPING THE CONTENTS OF MAG TAPE TO A TERMINAL
DUMPMT.SAV
DUMPMT.HLP
SPDUMP.FOR
SPDUMP.SAV
EBCDIC.DAT

PURPOSE: Both programs (DUMPMT & SPDUMP) have the same purpose. They both will write to a terminal, block by block, the contents of any magnetic tape. The tape can be labeled or unlabeled, and the character set can be either EBCDIC or ASCII. The maximum block size is 10240 byte characters.

The HLP document was written for DUMPMT, however, everything it says also holds for SPDUMP except :

1) SPDUMP uses only the FORTRAN callable routines that are standard in the RT-11 system. EXTMT is used by DUMPMT. EXTMT.MAC is a tape reading macro written by N A BOURGEOIS, JR. from the ADVANCED SYSTEM DEVELOPEMEMT DIVISION OF SANDIA LABORATORIES.
2) SPDUMP will tell you if there is any bad (unreadable) spots on the tape. It tells you where the error happened, then it automatically skips to the next tape mark. DUMPMT crashes in the same situation.
3) SPDUMP gives the option to either read all the blocks or just skip over them for each file that is skipped. DUMPMT will skip files, however while doing so it reads every block in the skipped file.

EBCDIC.DAT is the conversion file used by both programs to convert an EBCDIC tape into ASCII for display on a terminal.

****

FILES: BCK*.C HIGH BLOCKING DENSITY MAGNETIC TAPE BACKUP SYSTEM
BCK*.SAV
RET*.C
RET*.SAV
BACKUP.HLP

PURPOSE: This system can write to and read from a magnetic tape up to 100,000 (512 byte) blocks of data. A directory is first written to the tape. The data files are then dumped, back-to-back, on the tape. No tape marks are written between the files. They are all written to the same tape file at a blocking factor of 20.

The HLP file discribes what each program does and how they form a complete system. Command files can be used to execute the programs. Examples are given in the HLP documentation.

****

****

FILES: TAPER.FOR PROGRAM TO CREATE FSA DATA FILE FROM MAGNETIC TAPE FILE
TAPER.SAV
TAPER.HLP
TAPXXX.FOR
TAPXXX.SAV
EBCDIC.DAT

PURPOSE: The purpose of TAPER and TAPXXX are the same. They both will create FSA data files from files stored on a magnetic tape. They both also can read IBM EBCDIC tapes. The headers and trailers written on a labeled tape are handled as seperate tape files and can be skipped.

The HLP documentation was written for TAPER, but what is says also applies to TAPXXX except for the following:

1) TAPXXX uses only the FORTRAN callable routines that are part of the RT-11 system. TAPER uses the EXTMT.MAC.
2) TAPXXX reports any tape problem. TAPER will crash if there are any unreadable blocks.

EBCDIC.DAT is the conversion file used by these programs to convert EBCDIC into ASCII.

****

FILES: CRREAD.FOR SHARED FILE READING AND WRITTING

PURPOSE: CRREAD contains 4 subroutines which can be used by more than one user to simultaneously read and write to the same data files. These subroutines are:

1) FOPENS(fname,ibuf) ... Opens file 'fname', declares it to be a shaired file, and assigns it to channel 'ibuf'. Only channels 1 and 2 are allowed.
2) FCLOSE(ibuf) ... Closes files previously opened on channel 'ibuf'.
3) REDREC(ibuf,irec,lockon,lenrec,carec) ... Will read the 'irec' record on channel 'ibuf'. If 'lockon' is set to '1' the specific block which contains the desired record is also locked, therefore, prohibiting anyone else from accessing the same record. 'lenrec' is the length, in bytes, of each record in the data file. Once the specific record is read, it is placed buffer 'carec'.
4) RITREC(ibuf,irec,lockon,lenrec,carec) ... Will write the information located in buffer 'carec' to the 'irec' record on channnel 'ibuf'.

These routines use the functions which are available in the TSX-Plus system library (TSXLIB.OBJ). To operate these routines, your main program and subroutines must be linked to CRREAD.OBJ and to TSXLIB.OBJ.

```
>DIR MM0:[0,1],MM0:[1,2],MM0:[1,3],MM0:[1,5],MM0:[1,7],MM0:[1,6]

 Name .Ext    Size   Prot    Date        MM0:[0,1]
APL12 .RTS      88   <155> 20-Jun-83
APL16 .RTS      71   <155> 20-Jun-83
APL13 .RTS      88   <155> 20-Jun-83
APL17 .RTS      71   <155> 20-Jun-83

Total of 318 blocks in 4 files in MM0:[0,1]
```

Tapes

```
 Name .Ext     Size    Prot     Date        MM0:[1,2]
APLC12.APC        8    <155> 20-Jun-83
APLC13.APC        8    <155> 20-Jun-83
APLC16.APC       10    <155> 20-Jun-83
APLC17.APC       10    <155> 20-Jun-83
APL   .BAS        4    <155> 20-Jun-83
INVERT.APL        3    <155> 20-Jun-83

Total of 43 blocks in 6 files in MM0:[1,2]

 Name .Ext     Size    Prot     Date        MM0:[1,3]
CLOSE .APL        1    <155> 20-Jun-83
DUMP  .APL        1    <155> 20-Jun-83
TYPE  .APL        1    <155> 20-Jun-83
BYE   .APL        1    <155> 20-Jun-83
STORE .APL        1    <155> 20-Jun-83
SHOW  .APL        1    <155> 20-Jun-83
TIME  .APL        1    <155> 20-Jun-83
DATE  .APL        1    <155> 20-Jun-83
PRJPGM.APL        1    <155> 20-Jun-83
PRINT .APL        4    <155> 20-Jun-83
READ  .ME         1    <155> 20-Jun-83

Total of 14 blocks in 11 files in MM0:[1,3]

 Name .Ext     Size    Prot     Date        MM0:[1,5]
READFI.RST        3    <155> 20-Jun-83
PKVARS.HLP        2    <155> 20-Jun-83
PKRSTS.HLP        2    <155> 20-Jun-83
PCLN  .HLP        1    <155> 20-Jun-83
DPIPE .HLP        1    <155> 20-Jun-83
COPYA .HLP        1    <155> 20-Jun-83
XFNS  .HLP        1    <155> 20-Jun-83
PTO   .HLP        1    <155> 20-Jun-83
MOVE  .HLP        1    <155> 20-Jun-83
PKDIR .HLP        1    <155> 20-Jun-83
BYEF  .HLP        1    <155> 20-Jun-83
POFF  .HLP        1    <155> 20-Jun-83
PLOGIN.HLP        1    <155> 20-Jun-83
PKPFNS.HLP        1    <155> 20-Jun-83
PRJPGM.HLP        1    <155> 20-Jun-83
GCPIPE.HLP        1    <155> 20-Jun-83
CPY   .HLP        1    <155> 20-Jun-83
GCP   .HLP        1    <155> 20-Jun-83
PKLIB .HLP        1    <155> 20-Jun-83
RSTSM .HLP        1    <155> 20-Jun-83
SYSC  .HLP        1    <155> 20-Jun-83
PKINST.HLP        2    <155> 20-Jun-83
PSYSC .APL        1    <155> 20-Jun-83
CPY   .APL        2    <155> 20-Jun-83




PRINT .APL        4    <155> 20-Jun-83
PKPIPE.BAS       34    <155> 20-Jun-83
PKPPE1.CMD        1    <155> 20-Jun-83
NEWS02.SIG       35    <155> 20-Jun-83
PKNEED.           2    <155> 20-Jun-83
ARTCL2.CMD        9    <155> 20-Jun-83
RSTS  .APL       11    <155> 20-Jun-83
ARTCL2.LOG       73    <155> 20-Jun-83

Total of 199 blocks in 32 files in MM0:[1,5]
```

```
 Name .Ext     Size    Prot     Date        MM0:[1,7]
APL16 .RTS       71    <155> 20-Jun-83
APL17 .RTS       71    <155> 20-Jun-83
READ  .ME         1    <155> 20-Jun-83
APL17 .BAS        4    <155> 20-Jun-83
APL16 .BAS        4    <155> 20-Jun-83
APL   .BAS        1    <155> 20-Jun-83

Total of 152 blocks in 6 files in MM0:[1,7]

 Name .Ext     Size    Prot     Date        MM0:[1,6]
FORUTL.DOC       21    <155> 20-Jun-83
APLUTL.DOC       42    <155> 20-Jun-83
CHAREX.FOR        8    <155> 20-Jun-83
CHAREX.SAV       27    <155> 20-Jun-83
CHAREX.HLP        4    <155> 20-Jun-83
FIXLEN.FOR       10    <155> 20-Jun-83
FIXLEN.SAV       29    <155> 20-Jun-83
FIXLEN.HLP        3    <155> 20-Jun-83
FIXLEN.C          2    <155> 20-Jun-83
FSALEN.FOR        7    <155> 20-Jun-83
FSALEN.SAV       21    <155> 20-Jun-83
FSALEN.HLP        4    <155> 20-Jun-83
FSALEN.C          4    <155> 20-Jun-83
MERGEF.C         11    <155> 20-Jun-83
MERGEF.SAV       36    <155> 20-Jun-83
MERGEF.HLP       11    <155> 20-Jun-83
MATCH .FOR       20    <155> 20-Jun-83
MATCH .SAV       28    <155> 20-Jun-83
MATCH .HLP        7    <155> 20-Jun-83
MATCHB.FOR       22    <155> 20-Jun-83
MATCHB.SAV       29    <155> 20-Jun-83
MATCHB.HLP        7    <155> 20-Jun-83
UTLCAL.APL       10    <155> 20-Jun-83
UTLPRT.APL        4    <155> 20-Jun-83
REGRES.APL        7    <155> 20-Jun-83
UTLPRT.APD       20    <155> 20-Jun-83
UTILIT.APC       24    <155> 20-Jun-83

Total of 418 blocks in 27 files in MM0:[1,6]
```

ty dy1:readfi.rst

This account contains the complete PKPIPE interface to impliment a RSTS system link within an active APL workspace. New APL "commands" such as LIB, COPY, QUE, etc. are made possible. Also a RSTS function is provided to allow your APL functions to do anything you could do with a terminal at RSTS system level.

With that kind of flexible power you could write a package of very high level system management functions.

As described later, you must install one or more PKPIPE (s) on your system. The examples in this account and the command file expect certain things to be in place.

A log file - ARTCL2.LOG is provided to help you get started. After you have installed a PKPIPE according to the example, you can recreate the log file on your own system with ARTCL2.CMD via ATPK. This command file expects the multiprecision APL package in [1,3] of this tape to be installed prior to running it. It also expects files UTILIT.APC and UTLPRT.APD to be in [1,4] on your system, and all files from this account to be in the account you're logged into.

If you don't have a floating point processor, you cannot use the multiprecision APL package, but the ARTCL2 command file will do most of what it's expected to anyway.

Tapes

```
.ty dy1:*.hlp
 Files copied:
DY1:PKVARS.HLP to TT:
Important variables required in the RSTS workspace:

PKCHAN - A character variable containing a valid file channel
         number.  This is the channel used for the PKPIPE.
         eg. '10'

PWI    - Pipe Wait Index; normally set to 1 second.  All functions
         that set the input timeout duration of the PKPIPE should
         use this value as a base (eg. 2×PWI).  This allows the
         user to compensate for times of heavy system usage by
         increasing this value.

EOD    - End Of Data; normally 155.  This is the standard end of
         data byte value.  PKPIPE is normally set to return this
         byte after an input timeout.  This variable is defined
         by LOGIN.

DY1:PKRSTS.HLP to TT:
RSTS is the basic user level interface with the remote end of the
PKPIPE.  It requires a character right argument, though it may be
null.  If the argument is not null, RSTS sends the character data
down the PKPIPE, and using DPIPE, displays any response from the
pipe (see RSTSM).  If a null right argument is used, RSTS enters
simulated detach mode, supporting a two way conversation between
your terminal and the remote end of the PKPIPE.  RSTS is
transparent to most characters from the keyboard.  Some control
characters like ×Z and ×C however, have their usual immediate
effect.  Simulated detach mode is exited by entering APL as the
first three characters of a line.  If you find yourself caught
in a program at the remote end of the PKPIPE because you can't
send a ×Z or ×C in simulated detach mode, exit from RSTS with
APL, and use the SYS function to send a byte value of 3 for ×C
or 26 for ×Z.

DY1:PCLN.HLP   to TT:
PCLN (pipe cleaner) should be used if you get strange results
using a function that uses the PKPIPE.  Strange results can
be as drastic as a perpetual lockup condition.  This is probably
the worst thing that can happen, and is usually not as bad as
it looks.  Your function is probably receiving a timeout byte
value it doesn't recognize.  It will respond to ×C within one
pipe timeout interval.  Then you can use PCLN.  Remember to
CHECK YOUR STATE INDICATOR.

DY1:DPIPE.HLP  to TT:
DPIPE displays on your terminal, all bytes received from the PKPIPE,
until the pipe times out.  The pipe timeout byte value must be either
27 or equal to the variable EOD.

DY1:COPYA.HLP  to TT:
COPY, COPYA, and COPYD each expect as a left argument, a file
specification, and as a right argument, the names of one or
more objects in the specified file (workspace).  The left argument
should not specify the file type.  The arguments must be contained
in quotes.

DY1:XFNS.HLP   to TT:
XFNS, XFNSA, XFNSD, XVARS, XVARSA, and XVARSD list the functions
or variables in a stored workspace.  The right argument is a file
specification without a file type, and is contained in quotes.
```

DY1:PTO.HLP to TT:
PTO takes a scalar numeric right argument, representing the desired PKPIPE input timeout value in seconds. eg. PTO 3 will tell PKPIPE that if three seconds elapse without any data from the remote end of the pipe, it should give your program a pipe timeout byte to release it from it's input request.

DY1:MOVE.HLP to TT:
MOVE copies a file from one account to another and deletes the file from the original account. The left argument is the destination account in quotes, and the right argument is a file specification in quotes designating the file to be moved. The file name and type must both be specified, but can be ambiguous.

DY1:PKDIR.HLP to TT:
DIR, TYPE, and QUE are used like their CCL equivalents. The right argument is identical to what one would put after the CCL, except it's contained in quotes.

DY1:BYEF.HLP to TT:
BYEF and BYEY are used to terminate your slave job at the other end of the PKPIPE. Be sure you use one before terminating your APL session or the next person who uses that PKPIPE will have access to your files.

DY1:POFF.HLP to TT:
POFF makes sure you've opened a PKPIPE on the appropriate channel. If you have, it simply terminates. If you haven't it displays it's right argument on your terminal, and clears the state indicator. This protects other functions from trying to use a PKPIPE that isn't there.

DY1:PLOGIN.HLP to TT:
LOGIN logs in a job at the remote end of a PKPIPE. The job is created under your account. It also switches the slave (remote) job to the RSX run time system. LOGIN also establishes the Pipe Wait Index (PWI) at one second, and the pipe timeout byte to 155 (ASCII ESC with the high bit set). If you're using a privileged account, LOGIN sets the priority of your remote job to 0.

DY1:PKPFNS.HLP to TT:
Here's a complete listing of the APL functions required in the RSTS workspace. The display is in ASCII mnemonics, so if you're not used to these mnemonics, don't try to interpret them. Just use APL in TTY mode and enter them exactly as they're presented here.

DY1:PRJPGM.HLP to TT:
PPN and PRJPGM are utility functions that determine your account number.

DY1:GCPIPE.HLP to TT:
GCPIPE gets lines of character data from the PKPIPE. The explicit result consists of all received lines catenated together without carriage return or line feed characters. It takes two numeric scalar arguments. The right one indicates the number of lines to receive, and the left indicates the time in seconds to wait for each line before deciding to skip it.

DY1:CPY.HLP to TT:
CPY as a generic function, forms the major part of the COPY, XFNS, and XVARS groups. It is not used directly.

DY1:GCP.HLP to TT:
GCP is an ancillary function for GCPIPE.

Tapes

DY1:PKLIB.HLP to TT:
LIB is used like the APL)LIB command. It lists all files of type .AP? , and is used without arguments.

DY1:RSTSM.HLP to TT:
RSTSM (or RSTS More), simply invokes DPIPE. It's used when long delays in response from the remote end of the PKPIPE result in the pipe timing out before you receive all your data. This can happen if you use the RSTS function and the Pipe Wait Index (PWI) isn't set high enough. The effect is, you end up back at your APL prompt and data is still available in the PKPIPE. This is when you should invoke RSTSM.

DY1:SYSC.HLP to TT:
SYS and SYSC each take the right argument and send it to the remote end of the PKPIPE. The right argument for SYS is a numeric scalar or vector of ASCII byte values. This is useful for sending characters that are not recognized or disallowed by APL. The right argument of SYSC is a character vector or array.

DY1:PKINST.HLP to TT:

INSTALLING A PKPIPE ON YOUR SYSTEM

These instructions assume you're using pseudo keyboards KB8: and KB9: for your PKPIPE.

Include in your [1,2]TTY.CMD file, the following lines:

```
FORCE KB0: KB8:/RING:KSR33;TAB;LOCAL ECHO;SCOPE;NO UPARROW;WIDTH 132
FORCE KB0: KB9:/RING:KSR33;TAB;LOCAL ECHO;SCOPE;NO UPARROW;WIDTH 132
```

To have a PKPIPE automatically installed during system startup, add the following line to your START.CTL and CRASH.CTL files:

```
@[1,2]PKPIPE
```

and place a file called PKPIPE.CMD in [1,2] containing these lines:

```
DETACH
LOGIN KB: [1,2]
FORCE KB: RUN [1,2]PKPIPE
FORCE KB: KB8:
FORCE KB: KB9:
ATTACH
```

The following program "PKPIPE.BAS" should be compiled and placed in [1,2] with a protection code of 124.

Programming in VAX-11 APL

PROGRAMMING IN VAX-11 APL

Roger Matus
Digital Equipment Corporation

[Editor's Note: This is a copy of the slides Roger uses to talk about VAX-11 APL. This is all I could get on the subject from anyone. Roger indicated that the APL engineering staff is heavily engaged in projects at the moment and nobody at DEC has had time to write an article.]

STANDARD FEATURES

- All APL primitives including execute and domino
- Operators (reduce, scan, inner and outer product)
- System commands
- System functions and system variables
- File system

VAX/VMS APL COMMAND

- APL Command options:

  /HI /INPUT /SILENT /TERM

- Accepts initial workspace parameter:

  $ APL PAYROLL

- Initialization file: APL$INIT
- APL on-line VAX/VMS HELP module
- APL can be used from BATCH

PRIMITIVE EXTENSIONS

- Replicate
- Grade of characters and matrices
- Quiet functions
- Singleton extension

REPLICATE

o Compress accepts integer left argument

```
      1 2 ¯1 1 1 1/'FOBAR'
FOO BAR

      (1+S='''')/S          ⍝ DOUBLES QUOTES IN STRING S

      ((2×⍴S)⍴1 ¯1)/S       ⍝ DOUBLE SPACES STRING S
```

GRADE

o Works on characters and 2-dimensional arrays

```
      LANG
LISP
PASCAL
APL
      LANG[⍋LANG;]                LANG[;⍋[1]LANG]
APL                          ILPS
LISP                         ALAPCS
PASCAL                        PA L
```

SINGLETON EXTENSION

o One element vectors are treated as scalars

```
      ⍳1  1  1⍴4  ↔  ⍳4
```

QUIET FUNCTIONS

- o Quiet functions return a value only when needed
- o Eliminates the need to assign unwanted return values to supress output
- o Quiet functions include:

⊞ ⎕ARBOUT ⎕QCO

NEW SYNTACTIC FEATURES

- o Ambivalent user-defined functions
- o ◊ (diamond) statement separator
- o End of line comments
- o Whitespace preservation

AMBIVALENT FUNCTIONS

o Allows fns to be invoked monadically or dyadically

```
      ∇R←(A) MINUS B
[1]   →0=⎕NC 'A'/MONAD
[2]   R←A-B ◊ →0
[3]   MONAD: R←-B
      ∇

      6 MINUS 4
1
      MINUS 4
¯4
```

◊ STATEMENT SEPARATOR

o Provides a method of putting two statements on 1 line

o Allows advanced execute techniques:

```
⍎'X←1',(⍕N×11),⍞'◊X←X,+/¯2↑X'    ⍝ GENERATES FIBONACCI NUMBERS
```

END OF LINE COMMENTS

o Improves documentation and readability

```
      ∇ STAT X          ⍝ GET STATISTICS OF VECTOR X
[1]   (+/X)÷⍴X          ⍝ GET AVERAGE OF X
[2]   ⌊/X               ⍝ GET MINIMUM VALUE OF X
[3]   ⌈/X               ⍝ GET MAXIMUM VALUE OF X
      ∇
```

WHITESPACE PRESERVATION

o Tabs and spaces are preserved as typed in function lines

o Allows user-specified programming conventions

```
     ∇ A F B   ; L1 ; L2
[1] L1:     F X
[2] L2: G   X
[3] ⍬
     ∇ A F B   ; L1 ; L2
[1] L1:     F X
[2] L2: G   X
     ∇
```

EXPANDED ERROR MESSAGES

O Error messages now have secondary message in parenthesis

```
     ≠1 2 3
7 SYNTAX ERROR (NO MONADIC FORM OF FUNCTION)
     ≠1 2 3
     ↑
```

o Secondary messages can be supressed with ⎕TERSE

NEW DEBUGGING FEATURES

- o ⎕TRACE - set/show trace controls
- o ⎕STOP - set/show stop controls
- o Line 0 can be traced or stopped
- o STOP is trappable
  - Useful for performance and control flow analysis

SYSTEM COMMANDS

O	)INPUT	+	)OUTPUT
O	)PUSH	+	)DO
O	)LIB	+	)DROP
O	)FNS	+	)VARS

)INPUT

- Input files can be nested
- Character set can be specified

```
)INPUT FOO/TTY
```

-)INPUT/LIST shows nested input files
-)INPUT/REVERT clears all nested input files

)OUTPUT

-)OUTPUT/APPEND
-)OUTPUT/SHADOW - output goes to terminal and file
-)OUTPUT/REVERT
- Character set can be specified

```
)OUTPUT BAR/KEY
```

)PUSH

- Allows you to temporarily to leave APL

```
      )PUSH MAIL
MAIL>.
     .
     .
MAIL> EXIT

      BACK IN APL
```

)DO

- Captures the output of a DCL command

```
      T← ⍎ ')DO SHOW TIME'
      T
22-OCT-1982 11:19:31
```

)LIB

o)LIB and)DROP support full DCL functionality

```
)LIB/SINCE=1-JAN-1982/DATE/SIZE

Directory APLD$:[BLICKSTEI]

BAR.APL;1       14    06-OCT-1982 11:01
TAXES.APL;1     42    14-APR-1982 23:59

Total of 3 files, 56 blocks.

     )DROP/MODIFIED/SINCE=TODAY *.APL;*
%DELETE-I-DELETED, APLD$:[BLICKSTEI]BAR.APL;1 deleted
```

)FNS AND)VARS

o Now accept range specification

```
     )VARS
BRADY   CHIP   CHRIS   DAVE
DENNIS  ERIC   PETER   RICHARD
ROGER   STAN

     )VARS CHR R
CHRIS   DAVE        DENNIS  ERIC
PETER   RICHARD     ROGER
```

SYSTEM FUNCTIONS

⎕ARBOUT	⎕CIQ	⎕COQ
⎕NUM	⎕RESET	⎕TT
⎕FI	⎕VI	⎕QCO
	⎕QLD	

⎕ARBOUT

o Sends untranslated byte stream to terminal
- Can send any 7-bit character code
- Is not affected by current ⎕PW setting
- Useful for sending escape sequences and graphics commands

```
⎕ARBOUT 27 91 50 74      a (ESC)[2J  CLEAR VT100 SCREEN
```

⎕CIQ AND ⎕COQ

- Re-formats APL variables from one datatype to another
- Packs and unpacks data into integer arrays
- Allows you to combine heterogeneous arrays into one formatted integer variable

⎕NUM, ⎕TT AND ⎕RESET

- ⎕NUM provides a convienent character vector of numbers
  ⎕NUM ⟷ '0123456789'
- ⎕TT returns the current terminal type
  - Useful for writing terminal-independent applications
- ⎕RESET clears the SI stack
  - Handy for debugging workspaces

⎕FI AND ⎕VI

- ⎕FI converts character strings to numbers
- ⎕VI returns bitmap of valid numbers in string
- Useful for parsing complex input
- ⎕NG setting can be used to accept ASCII hyphen as minus

```
      T←'¯1 2 ¯2'
      ⎕FI T ◊ ⎕VI T
0 1 0 2
0 1 0 1
```

⎕QCO AND ⎕QLD

- ⎕QCO
  - Dynamically copies objects from saved workspaces via)COPY
  - Can be used to create an overlay mechanism
- ⎕QLD
  - Dynamically loads a workspace via)LOAD
  - Can be used to chain workspaces

SYSTEM VARIABLES

- ⍎ ⎕AUS controls automatic saving of CONTINUE ws
- ⍎ ⎕GAG allows or inhibits broadcasts
- ⍎ ⎕SINK provides an efficient method of discarding un-needed return values
- ⍎ ⎕SF quad-mode input prompt

```
      ∇ R←F;⎕SF
[1]   ⎕SF←'NUMBER PLEEZE: ' ◊ R←⎕ ∇
      F
NUMBER PLEEZE: 42
42
```

VAX-11 APL FILE SYSTEM

- o File organizations: ASCII Sequential, Internal Sequential, Direct-access, Relative
- o Complete RMS-32 integration
  - APL files are readable from other VAX Languages
  - All VAX/VMS RMS files can be read from APL
- o Mailboxes
- o Pure data modes: Integer, boolean, F and D floating, APL and ASCII text, and numeric byte
- o File system functions: ⍞, ⍞, ⎕ASS, ⎕DAS, ⎕DLS, ⎕DVC, ⎕CMS, ⎕FLS, ⎕MBX, ⎕RELEASE, ⎕CHANS

FILE ORGANIZATIONS: ASCII SEQUENTIAL

- o Reads and writes ASCII characters sequentially
- o Variable length records
- o Input and Output character sets: TTY, KEY and BIT
- o Input modes: ⍞, ⍞, ⍞
- o Shareable for input

FILE ORGANIZATIONS: INTERNAL SEQUENTIAL

- o Reads and writes APL objects or pure data sequentially
- o Variable length records
- o Shareable for input

FILE ORGANIZATIONS: RELATIVE

- o Reads and writes APL objects or pure data sequentially or via numeric record index
- o Variable length objects in fixed length records
- o Shareable for input and output via record locking

FILE ORGANIZATIONS: DIRECT-ACCESS

- o Reads and writes APL objects or pure data sequentially or via numeric key
- o Variable length records
- o Shareable for input and output via record locking
- o Records can be larger than 64Kb RMS record limit

EVENT FLAGS

- o Associated with I/O channel
- o Can be used independently of files
- o Event flag system functions:

  ⎕EFS ⎕EFR ⎕EFC

ERROR HANDLING

- o ⎕TRAP system variable contains string to be executed when an error occurs
- o ⎕SIGNAL generates an explicit signal in the callers domain
- o ⎕ERROR system variable containing message of last error
  - useful for determining cause of error and taking appropriate action
  - all APL errors are documented in the manual

ERROR HANDLING EXAMPLE

```
     ∇ R←A DIV B ; ⎕TRAP
[1]  ⎕TRAP←'→ERROR'
[2]  R←A÷B  ⍎  →0
[3]  ERROR: 'DIVIDE BY 0' ⎕SIGNAL 666 ∇
     3 DIV 0
666 DIVIDE BY 0
     3 DIV 0
     ↑
```

```
     ⎕ERROR
666 DIVIDE BY 0
     3 DIV 0
     ↑
```

NOTES

NOTES