the I.P.Sharp newsletter

May/June 1979
Vol. 7 Number 3

event TRAPPING

Automatic Trapping of Errors and Interrupts

Paul Berry, Palo Alto

SHARP APL now provides ways in which you can state in advance what you want the system to do when execution in your active workspace encounters an error or an interruption. The system may then respond automatically to an event which otherwise would have caused work to be halted.

An interrupt or error which can be intercepted is referred to as a **trappable event**. The term "event" is deliberately vague, partly to reflect the wide variety of things it covers, and partly to allow for the possibility that in the future the category may be expanded. It might then include some events that are neither errors nor interrupts, and do not halt execution.

When you provide an alternate instruction to be executed if a certain event occurs, you are said to **trap** that event. The SHARP APL event-trapping facility consists of the system variables □*ER* and □*TRAP*, and the system function □*SIGNAL*.

The text of this article is taken from the draft of "The SHARP APL Reference Manual", scheduled to appear later this year.

IN THIS ISSUE

Event trapping 1
RETRIEVE 2
SUPERPLOT 4
Stock Market data base 7
MAGIC with Bank data 7
Max-Pak 9
Phoenix office 9
APL80.....9
Recipes 10
Course schedule.....10
Technical Supplement 20
Event trapping I ..T1
Letters.....T6
Hardware moves T7
Fast rotate/reversal T8
Update: T8
(data base/public library)
Insert: plotted on a HP7221A
A. U.S. Crude Oil Stock
B. Timings on rotate/reversal

Some situations in which event trapping is useful

You can trap an error if (a) it doesn't happen every time, and (b) you're fairly certain of being able to provide a remedy when it does happen. In a situation like that, to verify in advance that no error will occur may be more difficult or more expensive than simply to let it happen and then correct it when it does. For example, if you're writing data to a file, there's a risk that there won't be room in the file to receive the next value. One approach would be to check in advance, each time you write a component, to see how much space is available and how much is required by your new data. Alternatively, you could set *⎕TRAP* so that **if** the file is full, some remedial action is taken (resizing it, repacking it, starting another, as appropriate). That way your program wouldn't need to check in advance.

A more striking (if specialized) example arises when you use matrix inversion. Some matrices are singular (have no inverse) and are rejected with the message *DOMAIN ERROR* when they're used in the right argument of ⌹. But to verify in advance that a matrix has an inverse is more expensive than matrix inversion. Using *⎕TRAP*, you could specify an action to be taken only if ⌹ has been attempted and has failed.

Without *⎕TRAP*, an application that accepts input from a user—especially from a user unsophisticated in programming—might require elaborate checking to verify that what the user has typed is acceptable. This, too, can be avoided by setting *⎕TRAP* to catch the errors that may arise, and then simply accepting input.

To make an application secure, you may want to prevent a user from interrupting some functions, so that the person at the keyboard has no opportunity to return to immediate execution and examine data that the function uses, files that are tied, and so on. By setting *⎕TRAP* appropriately, during critical segments of the program you can intercept the user's signals of "attention" or "interrupt," or use of *O* {backspace} *U* {backspace} *T* so that the person has no possibility of returning to immediate execution mode until you're ready to permit it.

Some events can't be trapped: Some kinds of interruptions can't be trapped. For example, a task can't trap a signal to bounce it. Some messages that you may receive at a terminal report events that aren't trappable. For example, you can't trap the trouble reports that arise from problems with a system command. And messages sent by the system's input processor, transmission control, or communications network—for example, *LINES DOWN* or *CHAR ERROR* or *PATIENCE PLEASE*—are completely outside the workspace, and not trappable.

(continued in the Technical Supplement)

RETRIEVE A Data Base Retrieval System

Marc Odho, Toronto

RETRIEVE enables you to retrieve data in standard format from a number of the public data bases available on the SHARP APL system. RETRIEVE is a time series data base retrieval mechanism — **not** a generalized data base management system. Typically, RETRIEVE is used in applications which require data for further processing such as portfolio management and time series analysis.

The retrieval mechanism is designed so as to be compatible with a wide variety of data bases. This means that even if your application requires data from several of the public data bases available on SHARP APL, you need only acquaint yourself with one method of retrieval. In addition, shadowing problems that occur while using retrieval functions with the same name are eliminated.

The RETRIEVE system refers to a data base as a **view**. Each data base, or view, consists of a collection of logically-related **facts** which are stored for one or more **objects**. For example, the view called *TSEINDEXHIST* contains data associated with the Toronto Stock Exchange Indices. Within this view, facts such as index value and dividend yield are stored for objects such as the gold index, the composite index, and so on.

Most of the data available using RETRIEVE is in the form of time series (i.e., a series of values or observations associated with a particular time interval or frequency). Facts are available with multiple frequencies. For example, a fact such as the volume of a stock trading on an exchange, can exist as a daily, weekly and/or monthly series. In addition, static facts such as names and descriptions associated with time series are also accessible using RETRIEVE.

Currently, historical Toronto stock exchange index data and Canadian/U.S. stock option market data are available via RETRIEVE. In the near future bond, commodity and equity data will be added.

A description of each of the views currently available using RETRIEVE is provided in the RETRIEVE View Directory (ask your local I.P. Sharp Associates representative), or on-line using the function *ΔVIEWDOC*. For a complete description of RETRIEVE refer to the manual "RETRIEVE - A SHARP APL **Data Base Retrieval System**".

In order to retrieve data using RETRIEVE, you have to provide the system with the four items of information outlined below. These items, or parameters, make it possible for RETRIEVE to uniquely identify any data item within a view.

1. To indicate which view contains the data you wish to access, you first specify a **view name**.

2. To indicate which facts within this view you wish to retrieve, you specify one or more **fact identifiers**

3. If the fact, or facts, you specify are stored for more than one object, you have to qualify your retrieval request with an **object identifier**.

4. If the facts being retrieved are time series, you qualify your retrieval request by specifying a **timeframe** of interest.

The view name, as well as fact and object identifiers, can be obtained from the RETRIEVE View Directory. Prior to performing an access, you would use the Directory to obtain the necessary parameter information for the retrieval in question.

For example, to retrieve the index value, dividend yield and names for the Toronto Stock Exchange Composite and Gold Index as of the first twelve months of 1976, you would proceed as follows:

```
ΔVIEW 'TSEINDEXHIST' {specify the
                      TSE300 view}
'MONTHLY' ΔTIME 197601 ΔTO 197612
                     {establish a time-
                      frame}
R←'TXTT,TXGL' ΔGET '1 4 11' {request
                      facts 1,4 and 11 be re-
                      turned for the objects
                      identified by the mne-
                      monics TXTT and TXGL
                      (i.e. the composite and
                      gold index)}
```

The result *R* is a package containing the required data in the variables *TSE*1, *TSE*4, and *TSE*11. If you specify *ΔRESULTTYPE 'GLOBAL'* prior to executing *ΔGET*, the variables containing the data (*TSE*1...) would be defined as globals in the active workspace.

Variables *TSE*1 and *TSE*4 are 3 by 12 arrays containing the index value and dividend yield. The first row contains the twelve month-end dates for 1976, the second and third rows contain the series for the composite (*TXTT*) and gold (*TXGL*) index, respectively. The variable *TSE*11 is a 2 by 43 character array—the first row contains the name for the composite index, the second row the name for the gold index.

RETRIEVE functions reside in workspace 55 *RETRIEVE*.

SUPERPLOT — A Really Super Plot Facility

Dave Keith, Toronto

Users of MAGIC and the public data bases available via MAGIC will be pleased to learn that a completely new version of the plotting facility in MAGIC—known as SUPERPLOT—is now available. This new version has several advantages over the current SUPERPLOT, including:

— **added support for the Tektronix 4015 screen and Tektronix 4662 flat-bed plotter, as well as for the Hewlett Packard 7221A flat-bed plotter.**

— **a dramatic increase in the style and type of plots available, such that SUPERPLOT is now a "state-of-the-art" system for time-series plots. The facility of the HP7221A to produce four-colour plots adds credence to the adage: "a picture is worth a thousand words".**

— **a decrease of approximately 20% in cost over the current SUPERPLOT for standard "default" plots.**

For the past two years, graphics style time-series plots have been available in MAGIC on hardcopy terminals only, under the name SUPERPLOT. On July 1, 1979, this software will be renamed *OLDSUPERPLOT* and the new package will become *SUPERPLOT*.

A conversational as well as a non-conversational system for the specification of the various options available with SUPERPLOT allows for the efficient use of this package by both novice and experienced programmers alike. Although the detailed settings to achieve each of the plots shown is not part of this article, a brief description of the method used for each is provided. Complete documentation for the new SUPERPLOT will soon be available from your local I.P. Sharp Associates office.

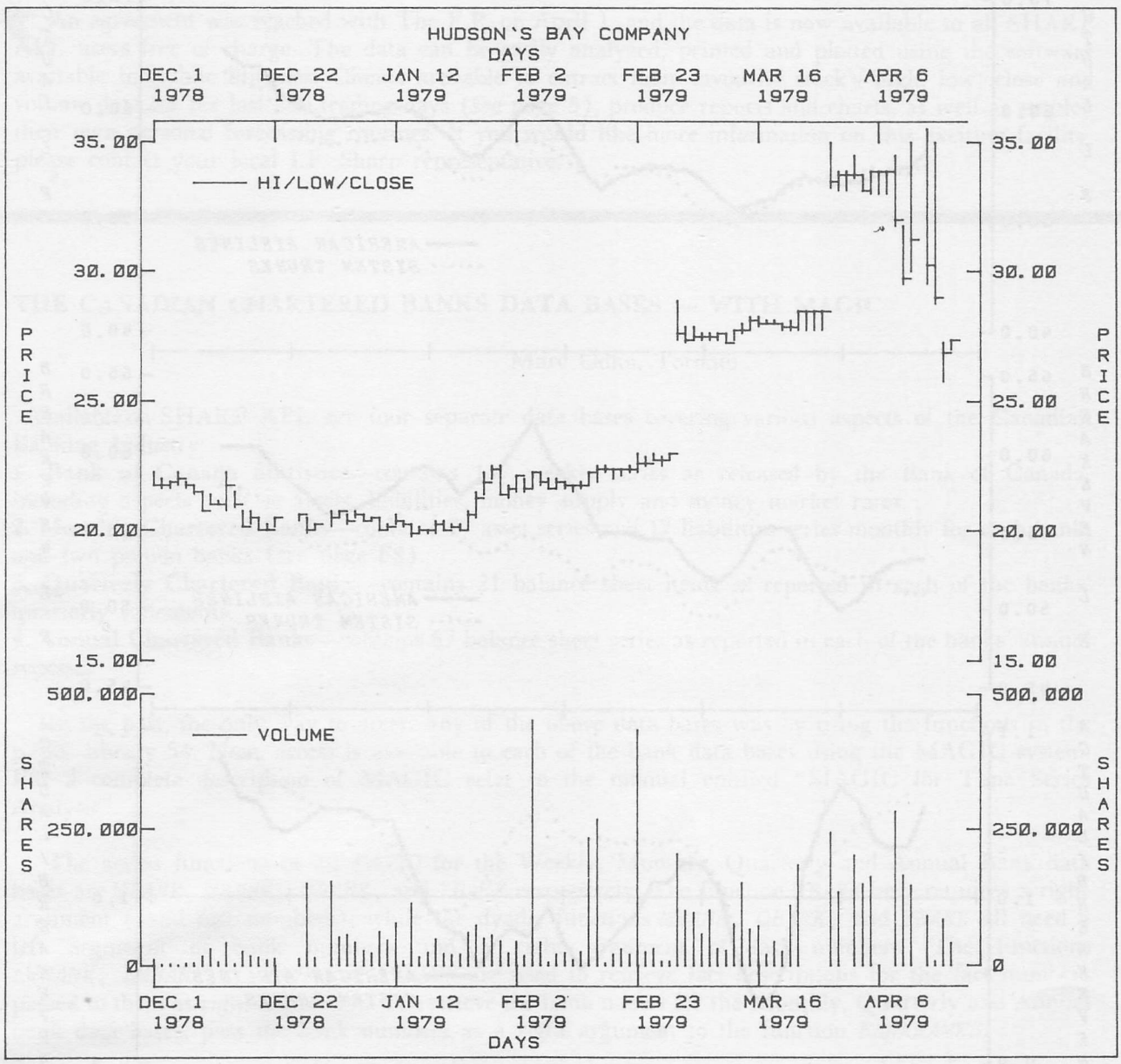
Most users of SHARP APL do not currently own a flat-bed plotter such as the Hewlett Packard 7221A (which was used to produce two of the three plots shown here). This reasonably priced, four-colour plotter is normally hooked up between the SHARP APL system and any standard ASCII terminal. Several I.P. Sharp offices are equipped with a flat-bed plotter, and interested users are welcome to test out the devices with their own plots. All of the features available with SUPERPLOT are available on either the flat-bed, screen, or hardcopy terminals. Users may even take advantage of two-colour ribbons on hardcopy terminals (e.g. black/red) to achieve two colour results. However, a word of caution is in order. Some features (such as shading) can be up to five times more expensive on hardcopy terminals, due to the fact that drawing a straight line between any two points requires the calculation and transmission of **all** points in between. On screens and flat-beds, the only actions required are the transmission of the start and end points, and the instruction to join them. Regular users of SUPERPLOT and other graphics packages will actually spend less money if they purchase the optimal plotting equipment.

Example 1 (INSERT A)

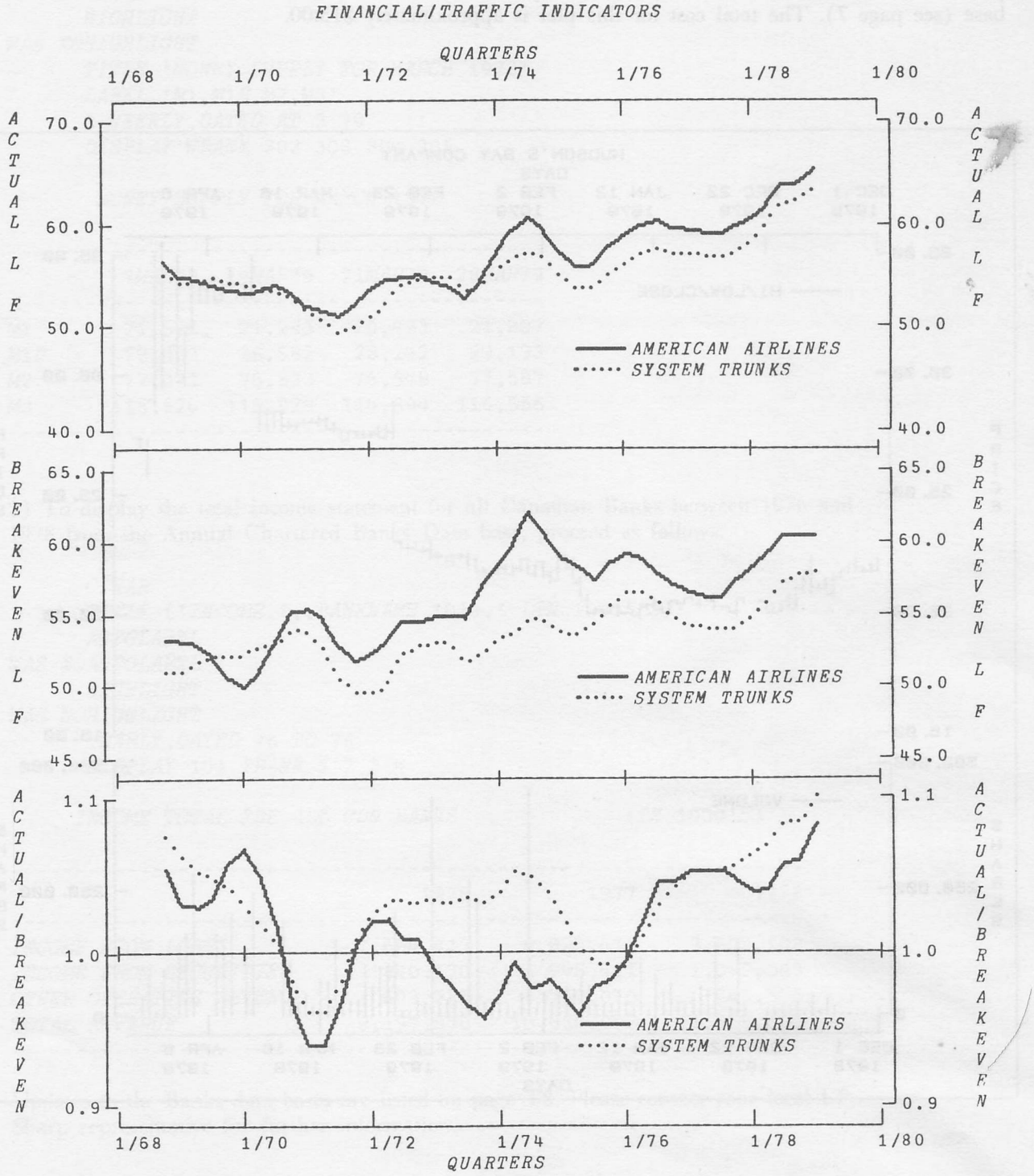
The first example, using data from I.P. Sharp Associates' new PETROSERIES data base was drawn on an HP7221A—**this plot can also be drawn on a transparent foil and used with an overhead projector.** It illustrates several features of the package. The heavy line in the top half of the split plot is actually drawn five times with a slight distortion for each pass. Shading at an angle of 45° from that line to the X-axis highlights this series. A histogram is selected as the line type for the bottom split, with very fine shading producing a solid effect. The percent changes are drawn relative to an artificial X-axis of zero.

Example 2

This plot, also a two-way split, illustrates two further line-type options—the high/low/close line type, and the vertical line histogram. Although not illustrated in this plot, several other SUPERPLOT features could have been employed here, including log scales; X and Y grids; varying character sizes for the title, legend and X and Y axes; double Y axes (e.g. the data for two companies on the same plot); and, of course, colour. The data for this plot is selected from the Financial Post Securities data base (see page 7). The total cost for this plot is approximately $15:00.



Example 3 Not to forget our old friend, the hardcopy terminal, this plot shows SUPERPLOT used in conjuction with the Sharp Aviation Data Base, in this case CAB Form 41 data. The bottom of the three splits shows actual load factor divided by break-even load factor. In cases where this ratio is greater than 1, an airline is making money, and in cases where it is less than 1, it is losing money. Different line styles are used in this plot to highlight one line against the other (solid versus dotted).



STOCK MARKET DATA BASE

Morgan Smyth, Toronto

We are pleased to announce that the access surcharge associated with the Financial Post Securities data bases has been eliminated. The data involved includes daily trading, statistics, earnings, dividends and indices for securities listed on the New York, American and Canadian exchanges. It also includes trading statistics for Trans Canada Options. Historically, access to this data was available to a select few. In order to use it, each client had to sign a separate agreement with The Financial Post, and pay an access fee each time they referenced the data, which tended to discourage those users who might only need access occasionally.

An agreement was reached with The F.P. on April 1, and the data is now available to all SHARP APL users free of charge. The data can be easily analyzed, printed and plotted using the software available in public libraries. Clients are able to extract their favourite stock's high, low, close and volume data for the last 260 trading days (see page 5), produce reports and charts, as well as employ their own personal forecasting routines. If you would like more information on this exciting facility, please contact your local I.P. Sharp representative.

THE CANADIAN CHARTERED BANKS DATA BASES — WITH MAGIC

Marc Odho, Toronto

Available on SHARP APL are four separate data bases covering various aspects of the Canadian Banking Industry:

1. Bank of Canada Statistics—contains 137 weekly series as released by the Bank of Canada, including aspects such as assets, liabilities, money supply and money market rates.

2. Monthly Chartered Banks—contains 27 asset series and 17 liabilities series monthly for each bank, and two pseudo banks (see page T8).

3. Quarterly Chartered Banks—contains 21 balance sheet items as reported in each of the banks' quarterly statements.

4. Annual Chartered Banks—contains 87 balance sheet series as reported in each of the banks' annual reports.

In the past, the only way to access any of the above data bases was by using the functions in the public library 54. Now, access is available to each of the bank data bases using the MAGIC system. For a complete description of MAGIC refer to the manual entitled "MAGIC for Time Series Analysis".

The access functions in `39 MAGIC` for the Weekly, Monthly, Quarterly and Annual Bank data bases are `WBANK`, `MBANK`, `QBANK`, and `YBANK` respectively. The function `WBANK` only requires a right argument (valid fact numbers), while the dyadic functions `MBANK`, `QBANK`, and `YBANK` all need a left argument of bank numbers and a right argument of fact numbers. The functions `ΔWBANK`, `ΔMBANK`, `ΔQBANK` and `ΔYBANK` are used to retrieve fact descriptions for the fact numbers passed to them as right arguments. To retrieve the bank names for the Monthly, Quarterly and Annual bank data bases, pass the bank numbers as a right argument to the function `BANKNAMES`.

Prior to performing an access, users should become familiar with the various facts and bank numbers necessary to identify the data—ask your local SHARP APL representative for the publications describing each data base. All identifiers used with the functions in library 54 are valid with their MAGIC counterparts, with the exception of `MBANK` where the codes `A` and `L` are not used. Instead, asset fact numbers are 1 to 27, and liability fact numbers are 101 to 117.

Examples

1) To display the Bank of Canada's set of measures of money supply (M1, M1B, M2 and M3) for March 1979—from the Bank of Canada's Weekly Statistics data base—proceed as follows:

```
      CLEAR
      HIGHLIGHT
WAS NOHIGHLIGHT
      TITLE 'MONEY SUPPLY FOR MARCH 1978'
      LABEL 'M1,M1B,M2,M3'
      3 WEEKLY,DATED AT 3 79
      DISPLAY WBANK 302 303 304 305

       MONEY SUPPLY FOR MARCH 1978

-----------------------------------------
         7MAR79  14MAR79  21MAR79  28MAR79
-----------------------------------------
M1       21,581   21,293   20,893   21,807
M1B      29,003   28,682   28,242   29,193
M2       77,071   76,833   76,548   77,587
M3      115,826  115,829  114,894  116,556
-----------------------------------------
```

2) To display the total income statement for all Canadian Banks between 1976 and 1978 from the Annual Chartered Banks Data base, proceed as follows:

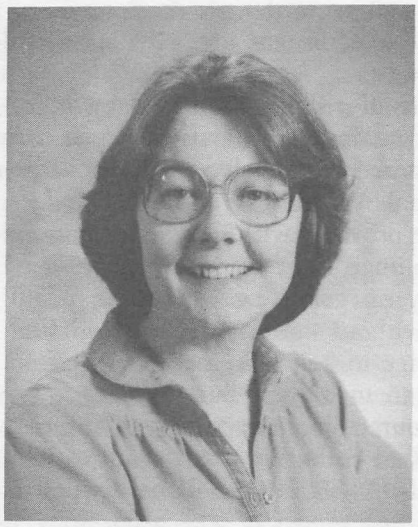
```
      CLEAR
      TITLE ('INCOME ',,BANKNAME 101),' (IN 1000''S)'
      AUTOLABEL
WAS NOAUTOLABEL
      HIGHLIGHT
WAS NOHIGHLIGHT
      YEARLY,DATED 76 TO 78
      DISPLAY 101 YBANK 1 2 3 4

    INCOME TOTAL FOR ALL CDN BANKS          (IN 1000'S)

-------------------------------------------------------------
                              1976         1977         1978
-------------------------------------------------------------
INCOME FROM LOANS        7,664,417    8,625,419    9,683,103
INCOME FROM SECURITIES     840,170      948,411    1,042,003
OTHER OPERATING REVENUE    693,286      757,070      843,571
TOTAL REVENUE            9,197,870   10,330,902   11,568,680
-------------------------------------------------------------
```

Updates to the Banks data bases are listed on page T8. Please contact your local I.P. Sharp representative for further information.

NEW BRANCH MANAGER IN PHOENIX, ARIZONA



Rose Mary Owen opened our Phoenix office in February, coming to Sharp with two-and-a-half years experience as a technical consultant for The Computer Company in Houston and Phoenix. She has an MA in mathematics from the University of Texas at Austin, and has a special interest in combinatorial mathematics, especially graph theory.

"How I got into APL? — about four years ago my brother suggested APL would be appropriate for implementing a project of interest to me then. I learned to use APL, and became fascinated with it. When I needed a job, I looked for an opportunity to use it. I especially like to find new, interesting applications. This past month, I finally found an application for the results of that project four years ago which started the whole thing!"

Rose Mary has four children, ages 7, 9, 15 and 17, and lived in New Mexico and Texas before moving to to Arizona in 1977. She holds an earlier degree in physics, taught school, and designed printed circuits. The address in Phoenix is: 3033 N. Central Ave., Phoenix, Arizona 85012, telephone (602) 264-6819.

An International APL Congress

APL/80

will be organized by the **Leiden University Computing Center** under the auspices of the **Netherlands Society for Informatics**

on
24, 25 & 26 June, 1980

Congress language: English

A call for papers will be mailed out shortly. For further information contact the organizing committee:

APL/80, c/o CRI,
Postbus 9512, 2300 RA Leiden,
The Netherlands.

Max-Pak
IS COMING!

Max-Pak, our **Microfilm Access Package** — combines the power and speed of computerized retrieval and economical microfilm storage. Data processing meets image processing, integrating the capabilities of Eastman Kodak, Data General and I.P. Sharp Associates. Max-Pak offers a new micro-image terminal to display document images, a computer to keep track of them (while keeping the cost down), and the software to tie the whole system together—a fully integrated turnkey system. The built-in flexibility of Max-Pak allows a wide variety of custom applications from the same standard package in such areas as banking, insurance, order entry, production and inventory control, law enforcement, personnel—any form of document control.

Another sharp idea.... watch for further details.

TOO MANY COOKS

Michael Berry, Boston

If a diet of aviation and financial statistics does not excite you, or if the gas and oil data seem a bit bland, cheer up. Sharp's newest database may be more to your taste!

95 *RECIPES* is a workspace designed to let you examine a file of recipes submitted by other gastronomically inclined users of the Sharp system. The workspace contains programs which you can use to enter new recipes, list out recipes, edit recipes you have entered previously, and even write reviews of the recipes you have tried. The programs are conversational. Whenever you do not understand a prompt, simply respond with a question mark and you will be provided with more information. Detailed documentation is available on-line, so don't be afraid to plunge right in.

There are several ways you can search for recipes: by name, by category (e.g. vegetarian), and by ingredients. You can combine these to search for, say, 'Italian, without tomatoes.' Once you find a recipe, you can have it multiplied by whatever amount is necessary to suit your needs.

Recipes have already been submitted for dishes ranging from flan to lobster, but we need a lot more in order to make the database truly useful. So please, share your culinary secrets with the rest of us as well as trying out the mouth-watering offerings of others. Real enthusiasts may want to join the new mailbox group *COOKS* where subjects of interest to users of 95 *RECIPES* will be discussed.

APL CLASSROOM

SEMINARS

Birmingham
Restart, May 25
Highspeed Print, June 29

Gloucester
Actuarial, August 20
APL Idioms, May 14
Box-Jenkins, June 18
Files, June 25
Financial, July 16
Linear Programming, September 17
MABRAUTIL, July 9
Project Planning, May 21
Report Formatting, September 10
Security, September 24

London
N- and B-tasks, September 6
Plotting with MAGIC/3 *PLOT/GRAFPLOT*, June 28
Your Bugs - How to get rid of them, May 22

Los Angeles
Data Base Design, May 15

Montreal
Financial Analysis, July 5-6
Plotting with SHARP APL, June 12

Rochester
APL Appreciation, June 7, June 21
Design of Files / Data Structures, June 9, July 11
Financial Planning, June 14
Forecasting with SHARP APL, June 28
Graphics, June 23
Report Formatting, June 26
Restartability / Reliability, July 12
Saving Money with N- and B-tasks, June 12

San Francisco
Graphics, May 9, July 18, Sept. 11, Nov. 6
MAGIC, June 20, September 19

Toronto
Actuarial APL Techniques, June 8, Aug. 13
AIDS - Introduction, June 14-15, Sept. 17-18
Box-Jenkins, May 8, July 9
Data Base Design, June 22, August 22
Forecasting Methods, May 11, July 11
MAGIC for Time Series Analysis, May 7, July 12
Plotting, May 10, July 19
Regression Analysis, June 13, August 16
Report Formatting, June 20, August 23
Saving Money with N- and B-tasks, June 4, August 2
Text Editing, May 28-30, August 27-29

Washington D.C.
Data Base Design, June 21 (morning)
Forecasting Methods, May 23
Petroseries, June 19 (half-day)

SPECIAL COURSES

'Appreciation of APL'
Gloucester, May 7, July 2, September 3
London, June 4, July 31, September 20
Warrington, May 16, September 19

'APL Review'
London, May 2, June 6, July 2, Aug. 8, Sept. 24

'Use a terminal'
London, May 11

ADVANCED APL

'APL and System Design'
Birmingham, May 14
London, May 24, July 19, Sept. 3
'Advanced APL'
San Francisco, August 7
'Advanced APL and Efficient Coding Techniques'
Toronto, August 20-21, November 1-2

INTERMEDIATE APL

Gloucester, August 13
London, May 3, June 7, July 3, Aug. 9, Sept. 25
Montreal, May 14-16, July 16-18
San Francisco, April 4, June 6, Oct. 9, Dec. 4
Stockholm, (in Swedish or English)
Toronto, May 22-24, July 23-25, Sept. 10-12

INTRODUCTION TO APL

Amsterdam
(Dutch)
May 16-18
July 18-20

Atlanta
(5 day)
May 7-11

Birmingham
(3 day)
May 8-10
July 10-12

Dallas
(4 day)
May 8,10,15,17
June 5,7,12,14
July 10,12,17,19
August 7,9,14,16

Gloucester
(3 day)
June 4-6

London
(3 day)
May 16-18
June 18-20
July 25-27
September 10-12

Montreal
(French/English)
May 7-9/22-24
June 4-6/19-21
July 9-11/24-26

Ottawa
(non-programmers)
June 5-7
(for programmers)
May 8-10
July 3-5

Rochester
May 21-25
June 18-20
July 23-27

San Francisco
(4 half-days)
May 22-25
July 10-13
August 21-24
October 2-5
November 13-16

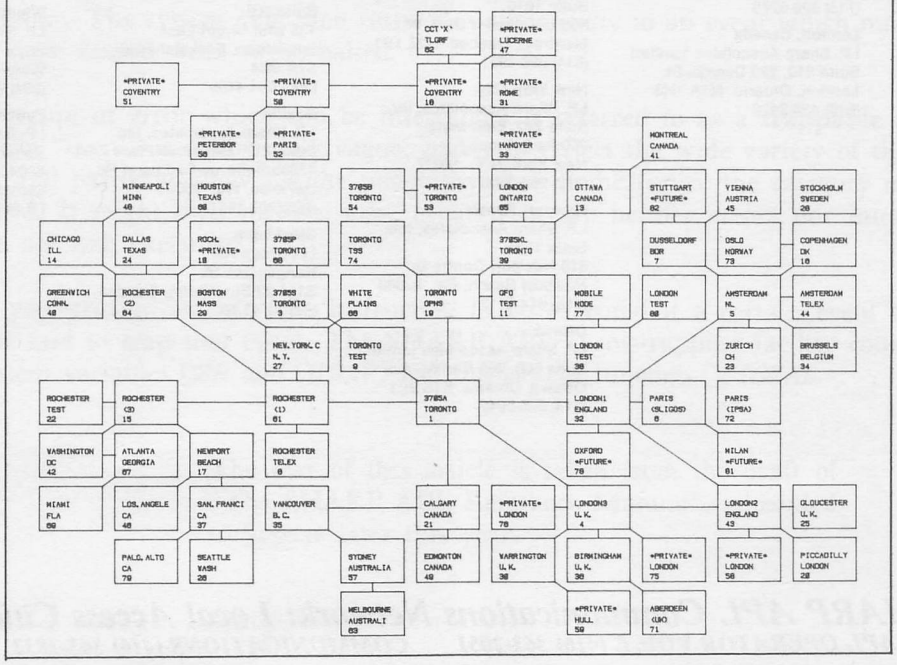
Toronto
(3 day)
May 14-16
June 5-7
June 25-27
July 16-18
August 6-8

Vancouver
(5 day)
May 23-25

Warrington
(3 day)
June 12-14

Washington
(5 halfdays)
May 21-25

Network Topology



UPDATE

☐ Please amend my mailing address as indicated.

☐ Add to your mailing list the following name(s).

☐ Send me a SHARP APL publications order form.

Name: _____

Co.: _____

Address: _____

I.P. Sharp Associates **Head Office:** 145 King Street West, Toronto, Canada M5H 1J8 (416) 364-5361

International Branch Offices

Aberdeen
I.P. Sharp Associates Limited
5 Bon Accord Crescent
Aberdeen AB 1 2DH
Scotland
(0224) 25298

Amsterdam
Intersystems B.V.
Herengracht 244
1016 BT Amsterdam
The Netherlands
(020) 24 40 50
Telex: 18795 ITS NL

Atlanta
I.P. Sharp Associates, Inc.
5000 Snapfinger Woods Dr.
Decatur, Georgia 30035
(404) 987-2301

Birmingham
I.P. Sharp Associates Limited
2nd Floor, Radio House
79/81 Aston Rd. North
Birmingham B6 4BX
England
021-359-6964

Boston
I.P. Sharp Asssociates, Inc.
Suite 812
148 State St.
Boston, Mass. 02109
(617) 523-2506

Brussels
I.P. Sharp Europe S.A.
Ave. General de Gaulle, 39
1050 Brussels, Belgium
(02) 649 99 77

Calgary
I.P. Sharp Associates Limited
Suite 2660, Scotia Centre
700-2nd St. S.W.
Calgary, Alberta T2P 2W2
(403) 265-7730

Chicago
I.P. Sharp Associates, Inc.
2 North Riverside Plaza
Room 1746
Chicago, Illinois 60606
(312) 648-1730

Cleveland
I.P. Sharp Asssociates, Inc.
Suite 590, 27801 Euclid Ave.
Cleveland, Ohio 44132
(216) 261-0800

Copenhagen
I.P. Sharp ApS
Østergade 24B
1100 Copenhagen K
Denmark
(01) 112 434

Dallas
I.P. Sharp Associates, Inc.
Suite 1148, Campbell Centre
8350 N. Central Expressway
Dallas, Texas 75206
(214) 369-1131

Düsseldorf
I.P. Sharp GmbH
Leostrasse 62A
4000 Düsseldorf 11
West Germany
(0211) 57 50 16

Edmonton
I.P. Sharp Associates Limited
Suite 505, 10065 Jasper Ave.
Edmonton, Alberta T5J 3B1
(403) 428-6744

Gloucester
I.P. Sharp Asssociates Limited
29 Northgate St.
Gloucester, England
0452 28106

Houston
I.P. Sharp Associates, Inc.
Suite 925, One Corporate Square
2600 Southwest Freeway
Houston, Texas 77098
(713) 526-5275

London, Canada
I.P. Sharp Associates Limited
Suite 510, 220 Dundas St.
London, Ontario N6A 1H3
(519) 434-2426

London, England
I.P. Sharp Associates Limited
132 Buckingham Palace Rd.
London SW1W 9SA
England
(01) 730-0361

Los Angeles
I.P. Sharp Associates, Inc.
Sherman Terrace Bldg.
18040 Sherman Way
Suite 118
Reseda, Ca. 91335
(213) 343-4617

Melbourne
I.P. Sharp Associates Pty. Ltd.
36 Elizabeth Street
South Yarra
Victoria, Australia 3141
(03) 244-417

Miami Lakes
I.P. Sharp Associates, Inc.
Suite D, Kennedy Bldg.
14560 N.W. 60th Avenue
Miami Lakes, Florida 33014
(305) 556-0577

Milan
I.P. Sharp Srl
Via Eustacchi, II
20129 Milan
Italy
(2) 271 6541/221 612

Minneapolis
I.P. Sharp Associates, Inc.
Suite 1371, 1 Appletree Square
Bloomington, Minn. 55420
(612) 854-3405

Montreal
I.P. Sharp Associates Limited
Suite 1610
555 Dorchester Blvd. W.
Montreal, Quebec H2Z 1B1
(514) 866-4981

New York City
I.P. Sharp Associates, Inc.
Suite 242, East Mezz.
200 Park Avenue
New York, N.Y. 10017
(212) 986-3366

Newport Beach
I.P. Sharp Associates, Inc.
Suite 1135
610 Newport Centre Dr.
Newport Beach, Ca. 92660
(714) 644-5112

Ottawa
I.P. Sharp Associates Limited
Suite 600, 265 Carling Ave.
Ottawa, Ontario K1S 2E1
(613) 236-9942

Oslo
I.P. Sharp A/S
Krønprinssesse Martha Plass 1
Oslo 1, Norway

Palo Alto
I.P. Sharp Associates, Inc.
Suite 201, 220 California Ave.
Palo Alto, Ca. 94306
(415) 327-1700

Paris
Société I.P. Sharp SARL
Tour Neptune - Cédex No. 20
92086 Paris-la-defense
France
(1) 773 57 77

Philadelphia
I.P. Sharp Associates, Inc.
Suite 407, 1420 Walnut Street
Philadelphia, PA. 19102
(215) 735-3327

Phoenix
I.P. Sharp Associates, Inc.
3033 N. Central Ave.
Phoenix, Arizona 85012
(602) 264-6819

Rochester
I.P. Sharp Associates, Inc.
1200 First Federal Plaza
Rochester, N.Y. 14614
(716) 546-7270
Telex 97-8380

San Francisco
I.P. Sharp Associates, Inc.
Suite C415, 900 North Point St.
San Francisco, Ca. 94109
(415) 673-4930

Saskatoon
I.P. Sharp Asssociates Limited
Suite 208
135 21st Street East
Saskatoon, Saskatchewan
S7K 0B4
(306) 664-4480

Seattle
I.P. Sharp Associates, Inc.
Suite 217, Executive Plaza East
12835 Bellevue-Redmond Rd.
Bellevue, Wa. 98005
(206) 453-1661

Stockholm
I.P. Sharp AB
Kungsgatan 65
S111 22 Stockholm, Sweden
(08) 21 10 19

Sydney
I.P. Sharp Associates Pty. Ltd.
Suite 1342, 175 Pitt Street
Sydney, N.S.W., Australia 2000
(02) 232-5914

Toronto
I.P. Sharp Associates Limited
145 King Street West
Toronto, Ontario M5H 1J8
(416) 364-5361

Vancouver
I.P. Sharp Associates Limited
Suite 604, 1112 West Pender St.
Vancouver, B.C. V6E 2S1
(604) 682-7158

Victoria
I.P. Sharp Associates Ltd.
Chancery Court
1218 Langley Street
Victoria, B.C. V8W 1W2
(604) 388-6365

Vienna
I.P. Sharp Ges. mbH
Rechte Wienzeile 5/3
1040 Vienna, Austria
(222) 57 65 71

Warrington
I.P. Sharp Associates Limited
Paul House
89 - 91 Buttermarket St.
Warrington, Cheshire
England WA1 2NL
(0925) 50413/4

Washington
I.P. Sharp Associates, Inc.
Suite 307, 1730 K Street N.W.
Washington, D.C. 20006
(202) 293-2915

Winnipeg
I.P. Sharp Associates Limited
Suite 909, 213 Notre Dame Ave.
Winnipeg, Manitoba R3B 1N3
(204) 947-1241

Zurich
I.P. Sharp A.G.
Badenerstrasse 141
8004 Zurich
Switzerland
(1) 241 52 42

SHARP APL Communications Network: Local Access Cities

APL OPERATOR VOICE (416) 363-2051 COMMUNICATIONS (416) 363-1832

Local dial access is available in all locations listed above. The SHARP APL Communications Network also provides local dial access in:

• Ann Arbor • Buffalo • Coventry • Dayton • Des Moines • Detroit • Ft. Lauderdale • Greene (NY) • Greenwich (Ct) • Halifax • Hamilton • Kitchener • Liverpool • Manchester • Raleigh • Regina • Syracuse • White Plains (NY).

In the United States the SHARP APL Network is interconnected with the networks of TYMNET and TELENET to provide local dial access in more than 100 other cities.

The Newsletter is a regular publication of I.P. Sharp Associates. Contributions and comments are welcome and should be addressed to: Jeanne Gershater, I.P. Sharp Newsletter, 145 King Street West, Toronto, Canada M5H 1J8.

Jeanne Gershater, *Editor* Ginger Kahn, *Assistant Editor*

Technical Supplement – 20

EVENT TRAPPING — PART I — Continued from page 1

Outline of the trapping facilities

Event report: When any interrupt or error occurs, the system stores as the value of ⎕*ER*, a character matrix containing a description of the event. The description includes the error message that may also be displayed at the terminal.

Trap definitions: The variable ⎕*TRAP* is used to indicate what events you want to trap, and what you want done when they occur. The system expects ⎕*TRAP* to be a character array, containing any number of **trap definitions**. Each trap definition is a list of events (identified by number), together with the action to be executed when one of them occurs.

Whenever an interrupt or error is detected, the system searches through the values of ⎕*TRAP* for mention of the type of event that has occurred, and, if it finds one, carries out the action specified there.

User-defined errors: The function ⎕*SIGNAL* permits a user-defined function to abort its own execution and generate an error signal, in much the same way that the system may abort execution of a primitive function and report an error.

How interruptions are handled when you are NOT trapping them

While the system is executing a line of APL (whether it's a line just entered from the keyboard or a line within a defined function), work may be interrupted for any of a variety of reasons. When that happens, unless you've used ⎕*TRAP* to provide otherwise, execution halts. If you're working from a terminal, following most events the system displays an error message. (Regardless of whether you're using ⎕*TRAP*, if the event is one that could be trapped, the message that's displayed is also included in the value of ⎕*ER*.) Then the system waits for you to enter from the keyboard an instruction telling it what to do next.

If you're running an N-task or a B-task, there's no way to receive an error message and no way to indicate what should be done next. Work simply stops. The system saves the task's active workspace.

While you're working at a terminal, your response to an interruption may often (although not always) include the following three steps:

1. Decide why the event occurred, by examining the message the system displayed, and perhaps by performing other tests.

2. Take necessary steps.

3. Resume work. If you were executing a defined function, you usually resume work by entering a branch statement such as →*LABEL* or →⎕*LC*.

The instruction you store in ⎕*TRAP* will often (although not always) provide the same three kinds of activity.

Event reporting

Event numbers: To indicate which events you want to trap, and to identify an event when examining ⎕*ER*, each kind of event has an **event number**. When the system puts into ⎕*ER* a description of the last error or interrupt, it always devotes the first four characters of the top row to the event number.

⎕*ER* **describes the most recent trappable interrupt or error:** Following an error (whether an error detected by the system itself, or one generated by use of ⎕*SIGNAL*), or an interrupt, the system generates an **event report**. (Event 2001, return to immediate execution, is trappable but is neither an error nor an interrupt, and isn't recorded in ⎕*ER*.)

The system assigns the event report as the value of the system variable ⎕*ER*. ⎕*ER* is a variable that the system sets, but you use. After an error or interrupt you may consult ⎕*ER* to decide on an appropriate action; similarly, the instruction you write in ⎕*TRAP* may also consult ⎕*ER* to select an appropriate remedial action.

Like any variable, ⎕*ER* may be made local to a function. When the name ⎕*ER* has been localized, the ⎕*ER* that receives the event report is the one that is visible; that is, the most local one.

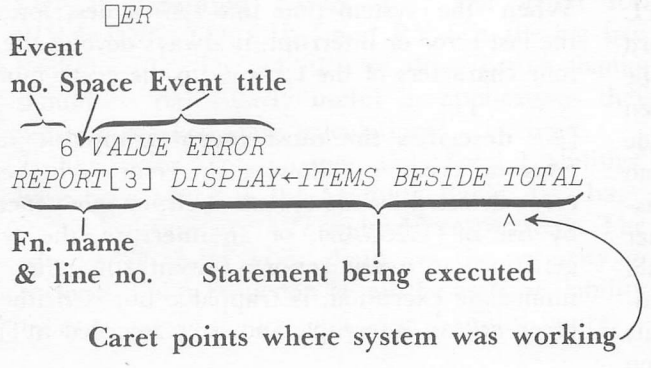
When the system specifies a new value for ⎕*ER*, any previous value of that ⎕*ER* is replaced by the new one. However, any more-giobal ⎕*ER* (shadowed by the local meaning) remains unaffected.

Since ⎕*ER* is a variable, you could, if you wanted, assign to it any value you like—but doing so would be pointless and possibly confusing. The system makes no use of ⎕*ER* other than assigning to it the latest event report each time an event occurs.

Form of ⎕*ER*: A value of ⎕*ER* set by the system is always a character three-row matrix. In a clear workspace in which no trappable event has occurred, ⎕*ER* has shape `3 0`. When the system sets a value for ⎕*ER*, it always includes the event number. If the workspace lacks space for a complete event report, the system makes ⎕*ER* a one-by-four matrix containing only the event number.

Event number: The first four elements of the first row of ⎕*ER* always contain the character representation of the event number, right-justified.

Suppose that while executing a function called *REPORT*, the system finds that a variable named *TOTAL* has no value. Here's how ⎕*ER* would appear:



Following the event number, ⎕*ER* usually contains a description of the event. The three rows of ⎕*ER* are used as follows:

1. Title (describing the event)

2. Statement (reproduced from the line being executed)

3. Caret (pointing to part of the statement)

When the event described in ⎕*ER* occurred during execution of a locked function, the system withholds some of the information that would otherwise be included. Row 2 of ⎕*ER* contains the function name and row number, followed by the character ∇. The caret on row 3 points to the ∇ symbol.

When the workspace lacks space to contain a complete ⎕*ER*, the system uses a terse form which contains only the four characters of the event number, as a one-by-four matrix.

Event title: Provided the workspace is not too full, following the event number the first row of ⎕*ER* contains one blank and then the **event title**. When the event was one recognized by the system, the system selects its title from its stock of standard phrases, such as *SYNTAX ERROR*, *WS FULL*, and so on. An event report generated by dyadic ⎕*SIGNAL* contains instead whatever title was specified in the left argument of ⎕*SIGNAL*.

Statement reproduced: The second row of ⎕*ER* reproduces the statement that was being executed when the event occurred. When that statement was part of a defined function, the function's name and the line number (in brackets), appear at the start of the second row. When the function was locked, the statement is replaced by the single character ∇.

When the event was an error generated by ⎕*SIGNAL*, the second row shows the statement that invoked the function whose execution was aborted by ⎕*SIGNAL*.

Caret: The third row of ⎕*ER* contains a number of blanks and a single ^ ("caret"). The position of the caret indicates where, within the statement reproduced on the second row of ⎕*ER*, the system was working at the moment the event occurred.

Default display: For almost all events, the message the system displays when you are **not** using □*TRAP* has the same appearance as □*ER*. However, in a default display, the system does not show the event number. For certain events (for example, halt before a line marked by a stop vector) the system doesn't display a title, but starts immediately with what's on the second row of □*ER*.

Classifying events to simplify trapping them

To make it simpler to specify which events you want to trap, events are divided into three classes:

1. Error: An error arises whenever the system finds that it cannot execute the line it is currently working on. An error may reflect an ill-formed statement (*SYNTAX ERROR*) or an inappropriate use of data (*DOMAIN ERROR*, *RANK ERROR*, *LENGTH ERROR*, *INDEX ERROR*).

An error may also arise when there is nothing wrong with the statement itself, but other conditions are not met. Lack of a needed object may cause *VALUE ERROR*; lack of space may give rise to *WS FULL ERROR* or *FILE FULL*, and so on.

Some error messages—such as *NO SHARES* or *FILE SYSTEM NOT AVAILABLE*—reflect a state of affairs over which you may have no direct control.

User-defined errors: Within a defined function, you can decide to abort execution and report an error, using the function □*SIGNAL*. Doing so halts execution of the statement that invoked the defined function, in much the same way as the system halts execution of a statement when it can't execute a primitive function.

When you cause a defined function to report an error, you may either report one of the errors that the system recognizes (such as *SYNTAX ERROR*, *DOMAIN ERROR*, etc.), or you may arbitrarily designate error conditions of your own.

Event number of an error: An error detected by the system has an event number in the range 1 to 499. To an arbitrary error report, signalled by □*SIGNAL* from within a defined function, you may assign any event number from 1 to 999. However, since the system uses numbers up to 499, it's prudent to confine user-generated errors to numbers in the range 500 to 999.

When you're specifying which events are to be intercepted by □*TRAP*, event number 0 is understood to mean "any error"—that is, any event whose number is in the range 1 to 999.

2. Interrupt: An interrupt is a signal which causes execution to be halted. In general, an interrupt is a signal received from outside the workspace. For example, signalling "attention" or "interrupt" while a function is being executed (by one or by two successive uses of the ATTN or BREAK key), or signalling "interrupt" during input (by typing *O* {backspace} *U* {backspace} *T*), is an interrupt.

A halt to execution caused by the way the STOP control (*S*Δ) is set is also considered an interrupt.

An interrupt is classified by what the system was doing at the time it occurred. Suppose you signal an interrupt from the terminal (by two successive uses of ATTN or BREAK). When the activity you interrupted was use of a shared variable, the system reports that as event 1005 *SV INTERRUPT*. But when the activity you interrupted was a file access, the system reports it as event 1006 *FILE INTERRUPT*.

Any of the current set of trappable interrupts can arise only from intervention by a user at a terminal (using the ATTN or BREAK key), or from the setting of the STOP control. The system function □*SIGNAL* **cannot** generate an interrupt.

Each interrupt has an event number in the range 1001 to 1999. When you're stating which events are to be intercepted by □*TRAP*, event number 1000 is understood to mean "any interrupt."

3. Return to immediate execution: Return to immediate execution is not an error or interrupt in the usual sense, and does not cause the system to set □*ER*, but **is** trappable. You can include in a trap definition provision to take certain actions if for any reason the workspace returns to immediate execution mode.

There's a return to immediate execution whenever the system completes execution of the last APL line entered from the keyboard (or when it completes execution of □*LX* after loading a workspace). There's also a return to immediate execution following any error or any interrupt which is not successfully handled by □*TRAP*.

When you're stating what events are to be intercepted by *⎕TRAP*, the event number 2001 is understood to mean "any return to immediate execution." Following return to immediate execution, the possible actions are (1) to do nothing; (2) to clear the workspace; or (3) to exit (see the description of action *D*). Trapping return to immediate execution serves primarily as a convenience at the normal end of an N-task or B-task, or as a security measure, to be invoked where other mechanisms have failed.

Trappable events: The system assigns an event number to each of the events it recognizes. A user-defined function using *⎕SIGNAL* can evoke any of the errors (having numbers between 1 and 499) or a user-defined error (having a number between 500 and 999). Event numbers 0 and 1000 are never directly reported by the system, but, when they appear in a trap definition's list of event numbers, serve as shorthand for a class of events. A complete list of event numbers can be found in SATN-32.

Events that are reported but not trappable

There are two error conditions which cause the active workspace to be cleared. The system displays the message *SWAP ERROR, CLEAR WS* or *SYSTEM ERROR, CLEAR WS*. Following these, the system sets *⎕ER* to show event 67 for *SWAP ERROR* and event 68 for *SYSTEM ERROR*. Since the active workspace is cleared, they couldn't possibly be trapped. They may be useful in understanding the fate of an N-task or B-task which is interrupted by one of these errors, since (unless *CLEAROUT* was set) the system saves its active workspace with *⎕ER* in it.

The form of *⎕TRAP*

Each *⎕TRAP* may contain any number of **trap definitions.** When a trappable event occurs, the system searches through the various trap definitions until it finds one that applies to the event.

⎕TRAP may be either a matrix or a vector. When it's a matrix, each row contains a trap definition.

When *⎕TRAP* is a vector, the system treats the first character as a **delimiter** which serves to separate the various trap definitions. Since the individual trap definitions will often contain APL statements to be executed, it's convenient and customary to use as delimiter a character that you're sure won't appear in any of the APL statements that the various trap definitions may include. In the examples that follow, the character ∇ is used as the delimiter, but any of several other characters would be equally appropriate.

The following example illustrates the two forms, matrix and vector. The content of the two is equivalent.

Matrix *⎕TRAP*:

```
      ⍴⎕TRAP
2 14
      ⎕TRAP
28 'E' →WAIT
18 24 'C' →RETIE
```

Vector *⎕TRAP*:

```
      ⍴⎕TRAP
29
      ⎕TRAP
∇ 28 'E' →WAIT ∇ 18 24 'C' →RETIE
```

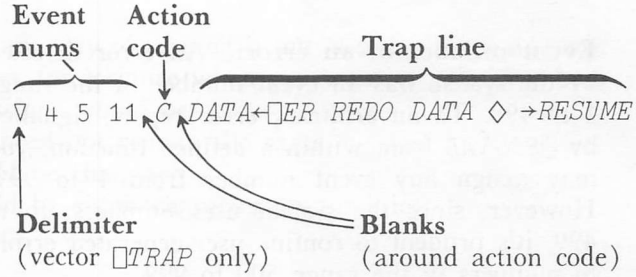
Form of a trap definition: Each trap definition contains at least two fields, and, optionally, a third. The three possible fields are:

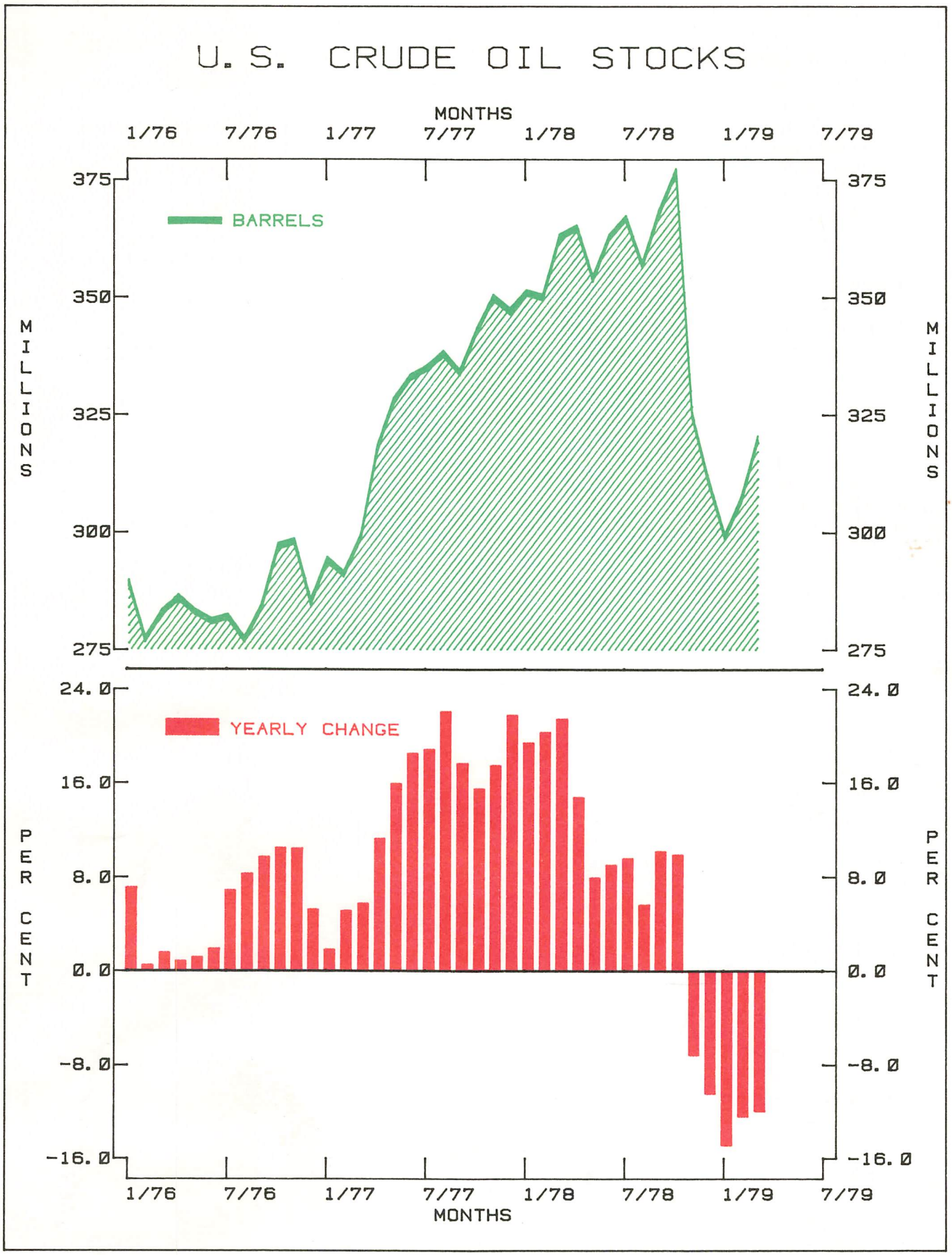
1. Event numbers: Lists (by number) the events to which the trap definition applies.

2. Action code: A single letter, separated from the adjacent fields by blanks.

3. Trap line or keyword (optional after certain actions)**:** Instructs the system what it should do following the event, and (optionally) how execution should be resumed.

Below appears a possible trap definition, to illustrate the various components just mentioned:




U. S. CRUDE OIL STOCKS
MONTHS
1/76
7/76
1/77
7/77
1/78
7/78
1/79
7/79
375
350
325
300
275
BARRELS
MILLIONS
24. 0
16. 0
8. 0
0. 0
-8. 0
-16. 0
YEARLY CHANGE
PER CENT
SUPERPLOT/I. P. SHARP ASSOCIATES

MAY JUNE/79

FIG 1
LAST AXIS 1 ROTATE
ON SQUARE MATRICES

SHARP APL TORONTO
TIME=04/21/79 21:11
USER LOAD=34
3 GRAFPLOT

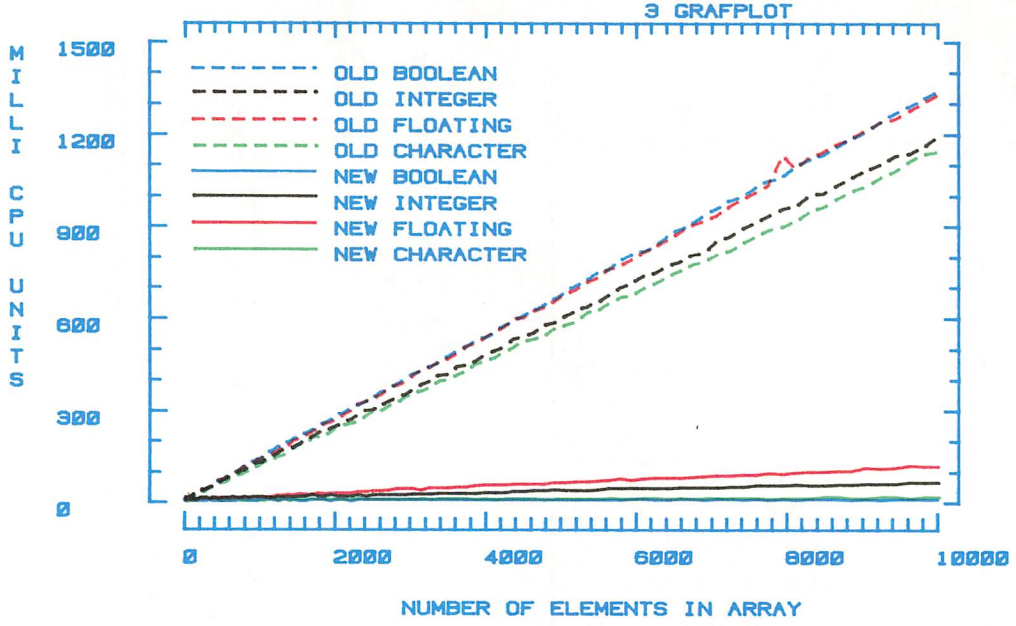


FIG 2
LAST AXIS REVERSAL
ON SQUARE MATRICES

SHARP APL TORONTO
TIME=04/21/79 21:34
USER LOAD=38
3 GRAFPLOT

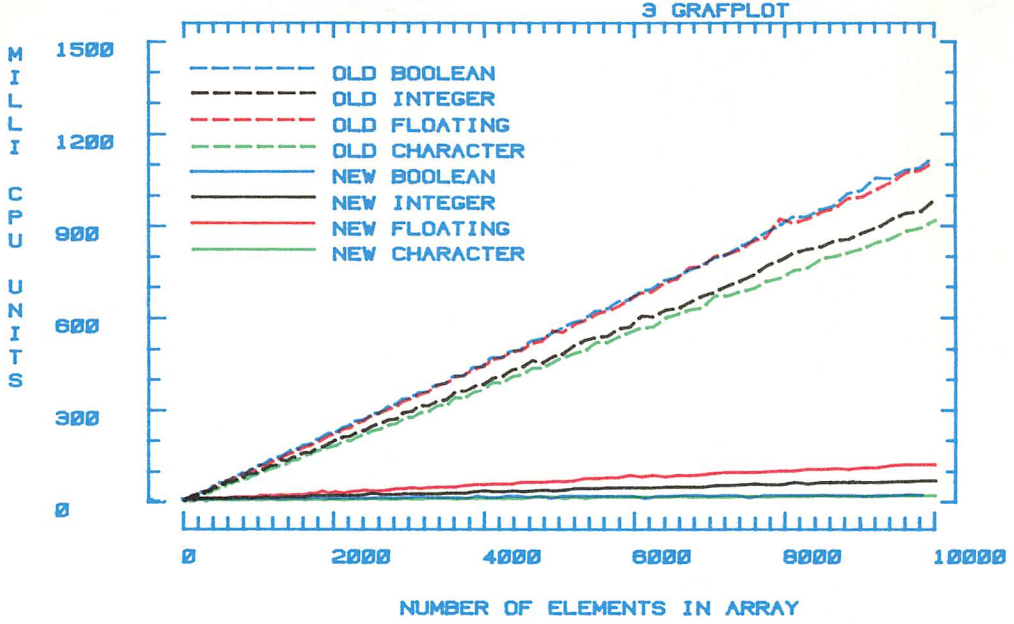


FIG 3
FIRST/LAST AXIS 1 ROTATE
ON SQUARE MATRICES

SHARP APL TORONTO
TIME=04/21/79 21:11
USER LOAD=34
3 GRAFPLOT

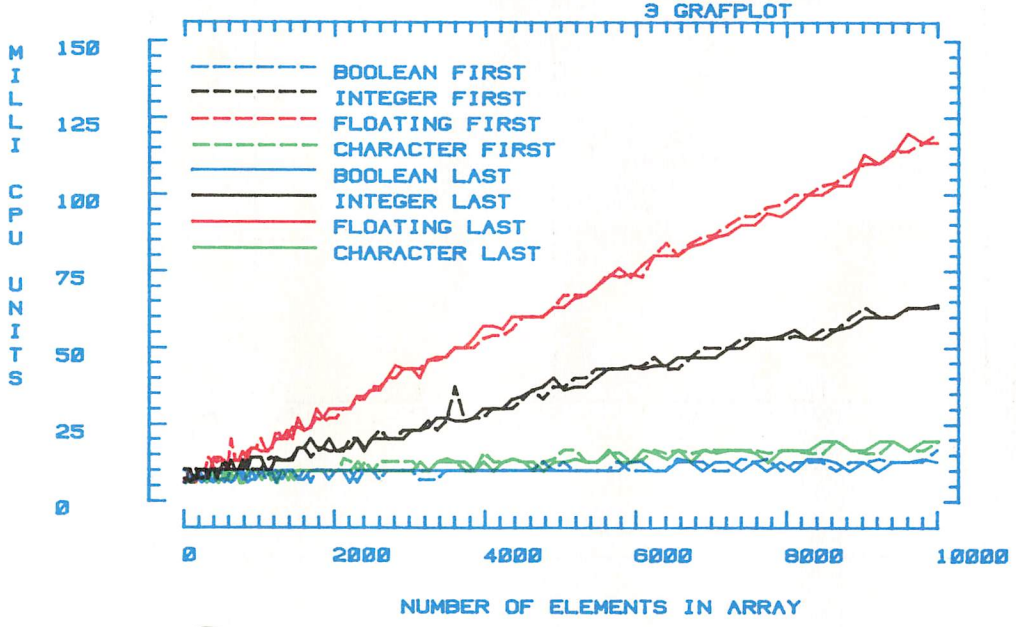
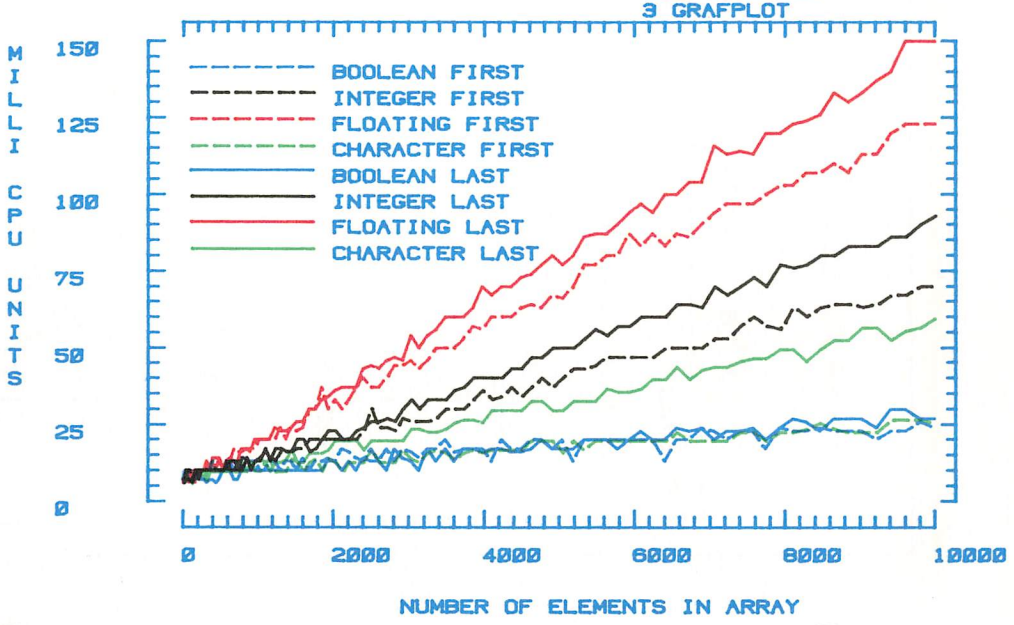


FIG 4
FIRST/LAST AXIS REVERSAL
ON SQUARE MATRICES

SHARP APL TORONTO
TIME=04/21/79 21:34
USER LOAD=38
3 GRAFPLOT



In the preceding example, the trap line contains two statements. The first respecifies the value of a variable *DATA* by invoking a function *REDO*, which refers both to the existing value of *DATA* and to *⎕ER*. The second instructs the system to resume execution by branching to *RESUME*.

System requires well-formed *⎕TRAP*: *⎕TRAP* is a variable you share with the system. Ordinarily, you set *⎕TRAP* and the system uses the value you've set. However, when you assign a value to *⎕TRAP*, the system requires that it be a well-formed set of trap definitions. If the value you propose contains a trap definition that is not well-formed, the system sets a new value for *⎕TRAP*, and makes it an empty vector. For example, you might observe the following sequence:

```
      ⎕TRAP←'JUNK'
      ⍴⎕TRAP
0
```

Criteria for a well-formed *⎕TRAP*: A well-formed *⎕TRAP* is one in which each trap definition is well-formed.

1. Each trap definition starts with the character representation of a non-empty list of event numbers.

2. Following the list of event numbers, and separated from it by at least one blank, each trap definition must contain one of the action codes *C E N S D*.

3. Following action code *C* or *E*, a trap definition may have a trap line. If there is a trap line, it must be separated from the action code by at least one blank.

Following action code *D*, there may be a keyword, which may be either *CLEAR* or *EXIT*. If there is a keyword, it must be separated from the action code by at least one blank.

Watch out: A frequent error in preparing a vector *⎕TRAP* is omitting the delimiting character. A delimiting character is required at the start of each definition, even when *⎕TRAP* contains only one definition.

A well-formed *⎕TRAP* isn't necessarily valid: After you set a value for *⎕TRAP*, it's prudent to examine it, to see whether the system has accepted it, or has reset it to an empty vector because it wasn't well-formed. But the fact that the system accepts a *⎕TRAP* as well-formed doesn't mean that the trap definitions it contains will work, or will do what you want. It means only that *⎕TRAP* is in acceptable form. When the system accepts a *⎕TRAP* as well-formed, it doesn't check the contents of each definition's trap line. An error in the trap line will become apparent when the system attempts to make use of it—that is, when it intercepts an error and attempts to execute the trap line. An error in the trap line is **not** trappable.

How the system locates the relevant trap definition

As soon as any trappable event occurs, the system halts work on the statement it was executing, and starts searching through the various trap definitions. If one doesn't cover the event that has just occurred, it goes on to the next trap definition.

Within a single value of *⎕TRAP*, it considers the various trap definitions one after another. When *⎕TRAP* is a matrix, it considers the trap definitions in sequence from top to bottom. When *⎕TRAP* is a vector with trap definitions separated by a delimiting character, it considers them in sequence from left to right.

As soon as the system finds a trap definition whose list of event numbers includes the event that has just occurred, it searches no further, and proceeds to carry out the action prescribed by the trap definition.

Search through multiple levels of *⎕TRAP* when *⎕TRAP* is localized: The variable *⎕TRAP* may be made local to a function. While that function is active, the local value of *⎕TRAP* is visible, and any other values of *⎕TRAP* outside the function are shadowed. You cannot see a shadowed value of *⎕TRAP*. **But the system can!**

While searching for a relevant trap definition, the system considers first the local, visible value of □*TRAP*. But if it doesn't find a match there, it goes on to consider each of the shadowed values as well. In this regard, the system's treatment of shadowed values of □*TRAP* is quite different from its treatment of shadowing in most other situations. (The system function `5 □WS` can show you at what levels of the state indicator □*TRAP* is localized.) For all other purposes, a shadowed value is simply invisible. But when the system is searching for a trap definition, when the visible value of □*TRAP* doesn't contain a relevant trap definition, it continues its search with the next shadowed □*TRAP*. If need be, it continues through all values of □*TRAP* that may exist at any level of localization.

Consequences of the way the system treats shadowed values of □*TRAP*

1. When you use a function with a local □*TRAP*, any event **not** covered in its trap definitions is governed by whatever trap definitions may exist in more global values of □*TRAP* (if any).

2. As long as a □*TRAP* exists, each of its trap definitions remains in effect **except** while a more local □*TRAP* covers the same event.

3. It's always possible to write a function so that trap definitions in its local □*TRAP* supersede any other trap definitions.

Watch out: You might think that by localizing □*TRAP* and making it empty, or failing to give it a value (so that there is **no** visible □*TRAP*), you'd interrupt event trapping. Not so. If your local □*TRAP* doesn't cover an event, the system simply looks at the next more global □*TRAP*—which you can't see but it can. Notice that your failure to specify a value for a local □*TRAP* has a consequence quite different from failing to specify the value of any other localized system variable, such as □*IO* or □*PW*. If you left one of those undefined but used a statement that required it, the system would signal a message such as □*IO ERROR*, □*PW ERROR*, etc., as appropriate.

(continued in Technical Supplement 21)

NEXT TIME:
- Actions in the trap definition
- Events that can't be trapped
- Signalling a user-defined error (□*SIGNAL*)
- Building and testing functions that use event

- Special precautions when trapping interrupts.

LETTERS

Application for □*TRAP*

I have used □*TRAP* to prevent errors causing abortive plotting actions when using the graphics package with the HP plotter. I have `□TRAP←'∇ 0 1000 E □ARBOUT 27 46 74 27 46 41◇□ER' ⍝27 46 41` halts plotting—`27 46 41` returns control to the terminal — so that when an error occurs in any plotting function the error message is not taken as stuff to drive the plotter.

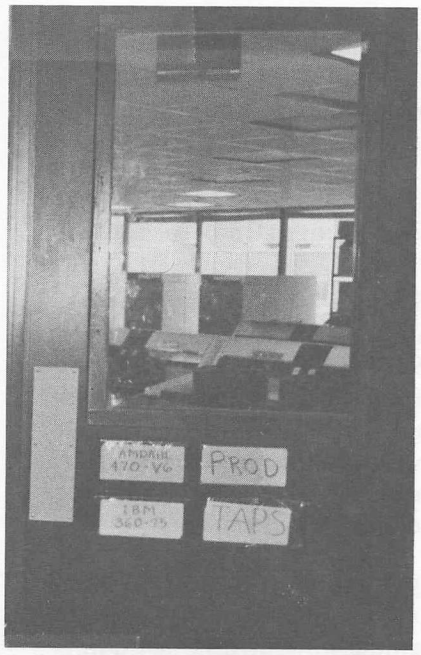
Bill Royds, Ottawa

Teeson's Teaser

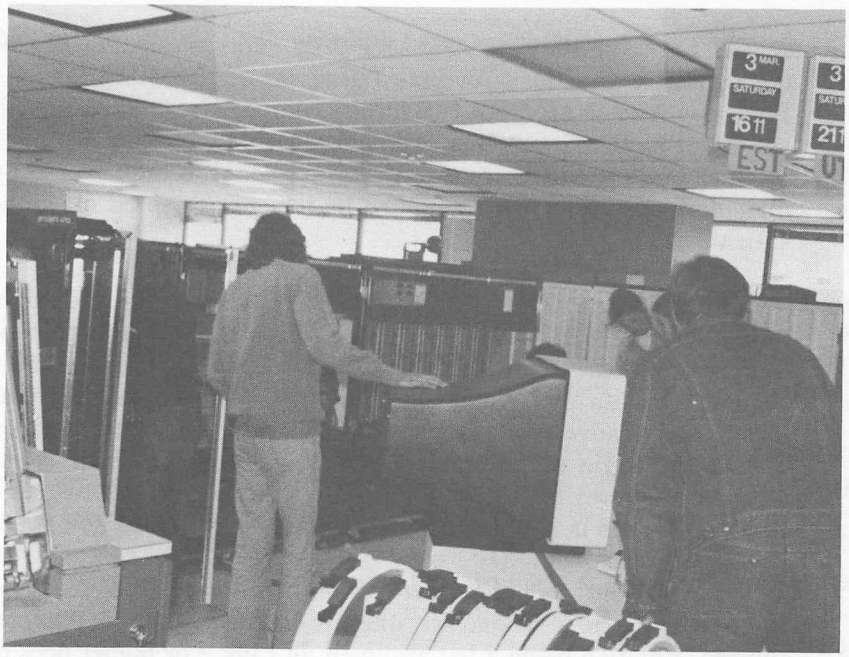
Given a numeric vector, *V*, what is the most elegant way of turning it into a one column numeric matrix of the same data type? This is often required, for instance when you want to print numbers down the side of a character matrix. Further extensions would be:
-how would your solution change if the data type need not remain the same;
-how would your solution change if the vector could be character;
-how would your solution change if the vector could be either character or numeric.

Peter Teeson, Toronto

TORONTO—**Installing the Second Amdahl 470/V6-II**



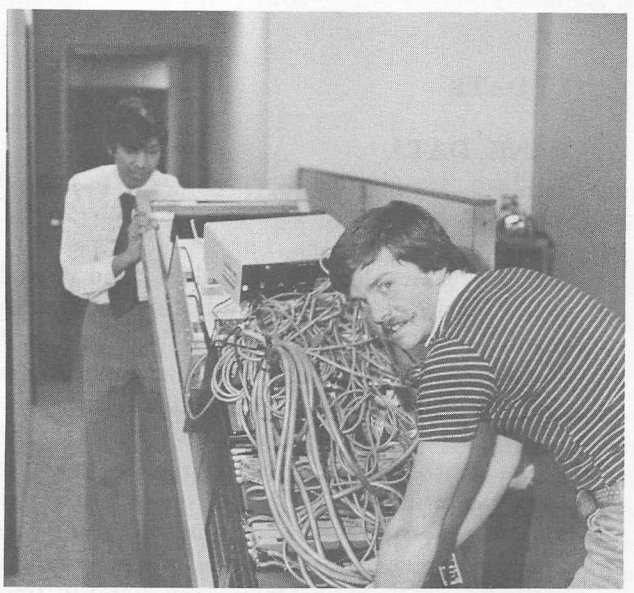
Taps for the last /75



High-powered Amdahl people completing installation of CPU 10062 (especially nimble-fingered **Norma Larosa** from California).

Roger Moore and Amdahl support folk search for the remote sense leads on the PDU.

MONTREAL—**Rohan Jayasekera** and **Dana Bradshaw** wrestling with the alpha during the move to larger offices.



... the first IBM computer to be installed in **China**, either this month or next, will be an APL machine. It will not have any other program languages on it—no Cobol or Fortran— only APL.

FAST REVERSAL AND ROTATE

Bob Bernecky, Toronto

On 14 April 1979, the SHARP APL Toronto system began using new algorithms to implement the primitive functions reversal (`Z←⌽[AX]A`, `Z←⊖[AX]A`) and rotate (`Z←B⌽[AX]A`, `Z←B⊖[AX]A`). Figures 1 and 2 (INSERT B) show the CPU time required by the old and new algorithms on a variety of matrix arguments. The old algorithms were not sensitive to data type. The new algorithms produce timings which are directly proportional to the size (in bytes) of the arguments, implying that their performance is largely limited by the storage speed of the host machine—in this case an Amdahl 470/V6-II.

The new algorithms perform somewhat better as the axis decreases. Figures 3 and 4 show timings for new reversal and rotate on square matrices. Note that reversal is more sensitive to the axis chosen than rotate, because reversal along the last axis has to move individual elements, whereas rotate along the last axis can move an entire row in two pieces. As the number of columns in the array becomes fewer, the relative cost per element will rise. The data in figures 3 and 4 is scaled up by a factor of 10 to highlight the differences in timing. Here the sensitivity to argument type becomes quite clear. The set of curves for reversal in figure 4 are first and last axis timings on square matrices, showing that CPU time is proportional to the axis over which the function operates.

Users should note that there is no measurable reduction in the CPU time required to perform rotation on vectors which are integer, floating point, or character, as these cases were already optimized. Rotation using a left argument of rank 2 or higher does not usually perform as well as rotation using a left argument of rank 0 or 1, because each element must be moved separately.

The algorithms detect certain cases which quickly produce the right argument unchanged as a result. These include reversal or rotation about an axis which contains 0 or 1 elements, and rotation by a scalar `A` such that `0=(⍴B)[AX]⌈A`.

UPDATE

BANK DATA:

1. Two pseudo banks have been added to the Monthly, Quarterly and Annual Canadian Chartered Banks data bases. The new "banks" are **Total for all Canadian Banks**—101, and **Total for** BOM, BNS, TD, CIBC **and** RY (i.e. the big five banks)—102. The addition of these pseudo banks should be particularly useful in applications that perform market share calculations.

2. The series 'Total Assets' and 'Total Liabilities' have been added to the Monthly Banks data base; their identifiers are A27 and L17 respectively. The addition of these series eliminates the necessity to perform a plus reduction on all the asset or liability series to arrive at the total asset/liability figure.

3. The function *WBDATES* in the workspace `54 WBANKS` now also returns weekly dates for integer week numbers (`1 - 10000`) passed as a right argument. Week number one (1) always returns the date of the most recent week's data. The previous use and syntax of *WBDATES* remains unchanged.

PETROLEUM DATA: Department of Energy monthly supply/demand data added. (Access via `121 PETROSERIES`.)

LIBRARY: Two workspaces, `51 STOCKOPTION` and `51 TSEINDEX` have been dropped, since the data is now available via `55 RETRIEVE` (see page 2).