the I.P. Sharp newsletter

Mar./Apr. 1979
Vol. 7 Number 2

SHARED VARIABLES

Why you might want to share one, and who you'd share it with

Paul Berry, Palo Alto

The SHARP APL implementation of "shared variables" is nearing completion. The work has been done by Dick Lathwell, who worked on the design and construction of the first shared variable system at IBM's Philadelphia Scientific Center, 1971-73.

What, you ask, is a shared variable? It's **one** variable which exists in **two** workspaces.

Ordinarily, what goes on in your active workspace is completely independent of what goes on in anyone else's. The APL system goes to elaborate lengths to isolate each active workspace from every other, so that no one else can know about or interfere with the work you're doing. Using shared variables, you can build a direct communications link between your active workspaces and another one that's running at the same time. Sharing is strictly by mutual consent; it takes place if and only if each of the workspaces makes the other a matching "offer to share" a variable (or several variables).

Once sharing is established, the variables that are shared between them exist simultaneously in **both** partners' workspaces. That provides a simple yet powerful means of communication. When one partner gives the shared variable a new value, the other one can read it. The act of setting a value for a shared variable has the effect of "send," while the act of referring to a shared variable has the effect of "receive."

IN THIS ISSUE

SHARED VARIABLES

Here's an example showing events in your active workspace before and after you agree to share a variable *X*:

Time	Your Active Workspace	Your Partner's Active Workspace
1	`X←⍳5`	
2	`X`	`X`
	`1 2 3 4 5`	`VALUE ERROR`
		`X`
		`∧`
3	**Sharing established between you and your partner**	
4	`X←'ABC'`	
5		`X`
		`ABC`
6		`X←2×⍳3`
7	`X`	`X`
	`2 4 6`	`2 4 6`

The shared variable facility includes several new quad-names. Here's a list of them, with a brief remark about what each is used for:

⎕SVQ SHARED VARIABLE INQUIRY asks what is the identification of the workspaces from which there are outstanding offers, or what variables a potential partner is offering you. Partners are identified by account number.

⎕SVO SHARED VARIABLE OFFER offers to share a variable of yours, either with a specific partner, or (if you want) with anyone — or inquires whether a variable is shared.

⎕SVR SHARED VARIABLE RETRACT withdraws your offer to share a variable. That ends sharing of that variable. Sharing also ends automatically if you erase the variable, or exit from a function to which it was local.

⎕SVC SHARED VARIABLE CONTROL lets you set (or inquire about) an interlock control for each shared variable. This is very important for assuring that communication doesn't get out of step. For example, if you want to send a succession of messages to your partner and get a reply from each, you want to be sure the partner reads each of your messages once and only once, and that you get one and only one reply. There's an independent control for setting and for reading by you and by your partner.

⎕SVS SHARED VARIABLE STATE lets you review who has given each shared variable a new value, and who knows what the values are. Especially when you're sharing several variables, perhaps with different partners, this lets you decide in advance which ones to read and which ones are ready to receive a new value.

⎕SVN SHARED VARIABLE IDENTIFICATION. This lets your workspace assign itself an additional identification, to distinguish itself from other workspaces running under the same account. It is important to do so when you're using N-tasks or B-tasks.

⎕SC STATE CHANGE. This permits a workspace (usually an N-task or a B-task) to "sleep" until one of the variables it's sharing receives a new value, or there's a new offer to share with it.

Shared variables provide a simple way to represent communication between any two things that most of the time work independently but from time to time communicate. A shared variable is what in engineering is called an "interface."

Who might you want to share with? In APL systems that lack SHARP APL's built-in file system, shared variables often provide the only way you can use files. In those systems, you offer to share a variable with a file processor (which looks like another APL user but is really a background program). You pass to it information about the files you want to read or write, and it passes back to you the information you've requested. That probably won't be a major use of shared variables with SHARP APL because direct file access is already built in, but it will permit you to use a style of communication that you may find in published programs that come from systems using that approach to files.

With SHARP APL, probably the most important use is to permit direct communication between the active workspace of your T-task with that of an N-task or B-task running at the same time. When you find part of a job you're running requires additional time or space, you can have it launch an N-task and remain in constant communication with it as it works.

Database applications are likely more and more to use a central workspace (probably an N-task or B-task) which runs over a protracted interval, and accepts share offers from many people at once. It manages their common files, while passing to each of its partners the results they've requested. Designing shared applications that way permits some major simplifications both in resolving races between users and in maintaining the system's security.

Detailed information on shared variables will be available in a SATN devoted to them, available shortly from your SHARP APL representative.

Caught in passing ....

Bernecky:
We were talking about nested arrays, which naturally resulted in Mike Jenkins and I (and Paul) talking about them as eggs. I suggested: "Sharp eggs on its customers...", whereupon Gene came up with: "..Sharp scrambles to the lead in General Array Derby ..." (headline in Datamation, July 1979..)

...later:
"Note that an enclosed array is a scalar which exhibits tree structure. A nested array is then an array of trees, which we may refer to as a forest. If we define ⍕ on arrays as Jenkins does (⍕ each leaf, then roll the thing flat), the *WS FULL* that occurs when you try to ⍕400 400⍴*ENCLOSE* ⍳1E3 is a classic example of not being able to see the forest for the trees.

IN-HOUSE MIGRATIONS - AN UPDATE

Ian Sharp

Some time ago we announced that the SHARP APL timesharing software was available for installation on customer mainframes. So far we have installed four such systems and the fifth and sixth are scheduled for April of this year. Two of the installed systems run on hardware dedicated to APL service: an IBM 370 model 158 and an ITEL AS6. One runs under VM in parallel with numerous other activities on an Amdahl V5, and one runs under MVT on an IBM 370 model 158. The next installation will also be under MVT on an IBM 370 model 165, followed by a second VM installation on an IBM 370 model 158.

Work is proceeding on the implementation of the MVS version and we have several firm commitments for it. We expect to make the first installation of SHARP APL under MVS late this year.

No firm orders have been received for an SVS version, and if this continues then no SVS version will be produced. If any customer does want one it would be scheduled to follow the MVS version and would be available in the second quarter of 1980.

Included with the SHARP APL timesharing software is software for the IBM 3705 which supports our packet-switched network protocol and effectively allows the customer's installation to be integrated into our communications network if the customer requires network services.

So far we have not experienced any serious technical difficulties and all installations have proceeded smoothly and on schedule.

All customers for timesharing software receive two releases per year containing all the new features of SHARP APL which have been operational on our timesharing service for at least three months. System programming help, should it be needed, is available 24 hours a day, 7 days a week.

Requests for further information should be directed to Joey Tuttle in Palo Alto.

APL79

Six papers by Sharp people were accepted for APL79 - the conference to be held in Rochester, N.Y. from May 30 through June 1. The titles and abstracts are:

R. C. Metzger, 'A Toolbox for APL Programmers'
A set of programming tools for APL programmers is described. One group of tools helps the reader understand APL programs by taking advantage of special features of communication terminals, as well as using nontraditional approaches toward program listings, and nonverbal communication methods. Another group of tools rewrite portions of APL programs in order to eliminate stumbling blocks to effective communication.

R. H. Lathwell, 'Some Implications of APL Order-of-Execution Rules'

An implementation of APL is necessarily a rigid encoding of the rules of the language, including a de facto order in which expressions are evaluated. The order of execution is perceived in an APL program by the way in which side effects such as nested specification, specification of global variables within defined functions, and shared variable values affect the outcome of expressions.

Concern about variation among implementations and distasteful results (see for example, "APL Problems with Order of Execution", Clark Weidmann, in the March 1978 issue of Quote Quad) led to a proposed definition for the order of evaluation of APL expressions which received general acceptance at an APL implementors' workshop at Minnowbrook in September 1977.

The new "right to left" evaluation rule is examined and compared with other possible choices, through the use of simple APL syntax analysis models, to determine its implications on storage requirements, and mathematical identities.

J. F. Sencindiver and D. H. Steinbrook, 'Multi-system Processing with APL'

Interfaces (both ASCII and EBCD) are presented which allow an IBM5100 or IBM5110 with APL to communicate as an intelligent terminal with a host APL timesharing system. The two concurrently active workspaces are coordinated through the Serial I/O Adapter Feature attached to a data set or acoustic coupler. Various transfer options are discussed, as well as hardware and software compatibility. A working application with cooperating workspaces is presented to illustrate some possibilities and implications of the interfaces.

E. McDonnell, 'Fuzzy Residue'

Certain pairs of arguments to the residue function, as implemented on many APL systems, give results which make it seem as if the ordinary decimal relationships we remember from grade school no longer hold. An algorithm for a fuzzed residue function is proposed which resolves the difficulties users have complained of. This proposal implies a related change in the definition of the representation function.

M. Dempsey, 'Using an APL Macroprocessor to Implement Generalized Software Systems'

Several techniques for implementing generalized software packages in APL are examined. One technique in particular, dynamic code generation, is expanded upon. When this technique is employed, application software is written in template form. These templates are then compiled into APL objects when an individual application is configured.

Some areas which are investigated are the use of templates, an APL macrolanguage, and an APL macroprocessor. Specific examples relevant to data base management systems are presented.

D. J. Keenan, 'Operators and Uniform Forms'

The syntax of operators is formally defined. This formalization permits an extension to current operators, and the introduction of new operators, two of which are proposed. The new operators, in conjunction with generalized uniform forms, are shown to be of great power allowing, among other things, easy manipulation of arrays of arrays.

There will be 54 contributed papers (in addition to nine invited papers) and several evening sessions. Contact Lynne Shaw, P.O. Box 40004, Rochester, N.Y. 14604, for details about the registration procedure.

GUEST PROCESSING WITH APL

David Steinbrook

The power of APL is now available at both ends of a phone line, using an interface between two APL workspaces. The case in which an IBM51X0 APL workspace is signed on to SHARP APL offers a powerful processing tool in itself, and acts as a model for extension of multi-processing techniques to systems with several SHARP APL tasks connected through the Sharp shared variable processor.

Three options currently available to an IBM51X0 (guest) workspace communicating through the serial I/O adapter feature with a T-task on the SHARP APL system (host) are:

1. Immediate processing controlled from the guest workspace:

∘ **exchange**: an APL expression, system command, or function-definition statement is passed from guest to host as (character vector) right argument to the guest-function *XCH*5, and the (character vector) result of host processing passed back to the guest workspace.

∘ **transfer**: five categories (APL object, group of objects, workspace, tape file, and disc file) may be transferred to or from either system.

∘ **nameclass processing**: a process requiring two workspaces can be split into stages by adopting conventions based on the nameclass of each row of a character matrix:
- a properly formed (and defined) name is regarded as an instruction to **transfer** both the object named and control of the processing of the next statement in the list to the other workspace.
- an improperly formed name is taken as an instruction to **execute** the statement in that row in the workspace currently assigned control.

Consider the namelist *NL*:

```
      NL
A←⍳8
A
B←A+3
C←5×B
C

      ⎕NC NL
4 2 4 4 0
```

Projecting the stages of this process onto two workspaces using nameclass control results in the following:

```
     Guest                        Host
A←⍳8        (↓EXECUTE)
A           (→TRANSFER)          A
            (EXECUTE↓)           B←A+3
            (EXECUTE↓)           C←5×B
C           (TRANSFER←)          C
```

Note that use of nameclass as the basis for selecting transfer or execution precludes display of a variable simply by naming it, or execution of a niladic function unless its explicit result is assigned to a name, or to ⎕ or ⍞.

2. Guest-to-Guest Processing. The Sharp shared variable processor allows a pair of host workspaces each serving as T-task to a guest workspace (G-task) to communicate directly. Access to a variable shared between guest and host (⍞) is coordinated with access to variables shared between T-tasks. Distributed data bases and distributed software may be updated, and APL models for dynamic resource allocation designed.

3. Deferred Processing. A scheduler allows requests for guest processing to be submitted on the host system by any SHARP APL user with access to this facility. Submission functions *GPROCREQ* and *AUTOGREQ* are analogous to functions used in submitting batch runs in SHARP APL. Processing specifications allow coordination of any guest processing request with any other, as well as with any batch task request. The guest processing facility is not restricted to the IBM51X0, and is designed to allow any intelligent processor (even if not APL intelligent) to be accessed indirectly from any SHARP APL task.

For further information please contact the author, or a SHARP APL representative in one of the 43 offices of I.P. Sharp Associates in North America, Europe or Australia.

GRADUATION

Jerry Cudeck, Toronto

Our actuarial offering has been enhanced by the addition of workspace `123` *GRADUATION*. This workspace contains a collection of functions which have been designed to aid in the process of securing from an irregular series of observed values of a continuous variable, a regular series of values, consistent in a general way with the originally observed series.

While traditionally falling within the domain of the actuary, the principles used in graduation theory extend immediately to meet the requirements of the statistician, economist, engineer, and others who routinely deal with curve fitting and interpolation.

The functions in the workspace have been designed in keeping with the concepts presented in the following two references:

Elements of Graduation, Morton D. Miller, Actuarial Monographs, No. 1 (The Actuarial Society of America), 1946.

Part 5 Study Notes, Graduation, T.N.E. Greville, Education and Examination Committee (Society of Actuaries), 1974.

Generally speaking, each function falls into one of the following categories:

1) Interpolation
2) Moving-Weighted-Average
3) Whittaker-Henderson
4) Fit and Smoothness Checks

A manual, "Graduation", is now available, in which each function is discussed in one of these categories. Function description, syntax, and examples of function usage are provided. The examples use tables which are present in the workspace. For further details, see *TABLESHOW*, *INTERPOLATIONHOW*, *MWAHOW*, *WHHOW*, and *FITHOW* (and for a full description, *DESCRIPTION*). Questions concerning this workspace may be directed to Jerry Cudeck or Dave Hosier in the Toronto office.

ENHANCEMENTS TO THE SHARP APL STATISTICAL LIBRARIES

Andrew North, Ottawa

Growth and development of the Statistical libraries continue with the generation of a new workspace and significant extensions to an existing workspace. The enhancements will be of interest to those users involved with multivariate analysis or with the seasonal adjustment of economic time series.

Multivariate Analysis: 35 *PRINCIPAL*

The workspace 35 *PRINCIPAL* contains functions to perform two types of multivariate statistical analysis: principal component analysis and canonical correlation analysis. Each of these techniques can be applied to obtain a more parsimoneous description of the variability existent in multivariate data sets. The principal component analysis finds linear combinations of one set of variables which together account for as much variation as possible within that variable set. The canonical correlation analysis finds linear combinations of each of two sets of variables with the aim of accounting for a maximum amount of the relationship between those two variable sets.

Each of these analyses can be performed based upon a correlation matrix or upon a covariance matrix, either of which is estimated from the input data. Output from each analysis includes the coefficients of the corresponding linear combinations, starting with those combinations which provide the largest explanation of the variability existent in the original data. The workspace also allows evaluation of the linear combinations at each of the original data observations, as well as estimation of the correlations between the original variables and the resulting linear combinations of those variables.

Seasonal Adjustment: the workspace 39 *X*11

39 *X*11 performs the seasonal adjustment of time series according to the X-11 variant of the Census Method II Seasonal Adjustment program. The algorithm, originally developed by the Economic Research and Analysis Division of the U.S. Department of Commerce, Bureau of the Census, purports to incorporate the optimum features of the various methods available for the adjustment for seasonality of economic time series.

The fundamental assumptions underlying the techniques are that seasonal fluctuations in the original series are not only measurable but are separable from the other components of the series, namely the trend, cyclical, trading-day and irregular components. The seasonal adjustment algorithm attempts to remove seasonal and trading-day variation, so that the final seasonally adjusted series consists of the trend-cycle and irregular components only.

The workspace 39 *X*11 performs such adjustment according to a variety of user-specified output and computational options, a number of which were not available in the previous release of the workspace. State setting functions control selection of such adjustment parameters and output options as the following:

- definition of a series title, series frequency and hypothesized model type (i.e., the manner in which the components of the series are assumed to be related).
- definition of the magnitude of the error bands used in the identification of extreme irregular values.
- incorporation of optional adjustments designed to take into account the effects of strikes, trading-day variation, prior information, and, for series concerning retail sales data, the calendar location of certain holidays.
- calculation and printing of summary measures tables.
- selection of those tables (of the 54 available) which are to be printed, and whether these tables are to be printed on the terminal or on the highspeed printer, or both (or neither).
- whether the adjustment is to be performed as a T-task, N-task or B-task, an execution cost reduction being effected by the latter two selections.

An additional new feature of the workspace is a facility by which a number of series can be adjusted as a single task submission. Adjustment parameters and output options can be altered for each individual series, as required. Such results as the final seasonally adjusted series are appended to a user-created file in the form of APL packages.

For additional information concerning this new facility and all other features of 39 *X*11, see the revised version of the manual "X-11 in SHARP APL", now available from your SHARP APL representative.

SHARP NEWS

Saskatoon: Allison Atkey has established a new office, at:
135 21st Street East,
Saskatoon, Saskatchewan, S7K 0B4
(306) 664-4480

Phoenix, Arizona: Rose Mary Owen is opening a new office in the Southwest. She can be reached at:
1245 E. La Jolla,
Tempe, Arizona 85282.

Oslo, Norway: local dial access is available.

Technical Supplement–19

FUNCTIONS AND OPERATORS

Joey Tuttle, Palo Alto

I have for a long time considered APL a thought discipline and as such have become increasingly interested and aware of the reasons for its usefulness and power. I have come to believe very strongly that the primary reason people find APL so useful is its solid basis in mathematics. APL has made several contributions to mathematical notation and problem solving methods. For example, the floor and ceiling functions (⌈ and ⌊) are extensions of the idea of the "greatest integer function" of conventional mathematics. Also, APL has removed ambiguities such as confusion between the subtraction function and the symbol used to indicate a negative number. APL has also provided consistent definition of the entire set of relational functions as distinct from the idea of assignment of a value to a name.

A further contribution of APL to other standard notations has been the idea that "primitive monadic functions" always take as their argument the value of the expression to their right. An example is the increasing use in general publications of |*A* to indicate the absolute value of *A*, instead of the more traditional |*A*|. The point of all this is that concise, consistent, and general ways of expressing ideas are extremely important aspects of APL. These aspects lead to the importance of studying and understanding the semantics of APL.

The "order of execution" of APL can be simply and elegantly defined by saying "All functions act on the value of the expression to their right and may in addition use the value of the expression immediately to their left if they are dyadic." This consistency removes the need for a hierarchy of functions in APL. The key point here is that **functions** always take **values** as their arguments, and (should) in turn return values as a result.

An idea which has been embodied in APL from its beginnings, but has only recently been properly appreciated, is illustrated by the APL expression for summation (+/*V*). It has become clear in the past few years that the best way to explain the summation expression is to consider the / as an "operator" which follows a different set of rules than functions. The reduction operator (/) is a monadic operator and takes as its argument the **function** +. Note that the value *V* is not an argument of the operator. The action performed by the reduction operator is to create a "derived function" called summation. The derived function is monadic, takes the value *V* as its argument, and returns the value of the sum of the elements of *V*.

In a similar manner the dyadic "product operator" (.) takes as its arguments the functions to either side and returns a derived function called an "inner product". The product operator can be used with the place holder ∘ to obtain a monadic form that is the basis for the "outer product" functions (which are very useful in generating tables).

The derived functions generated by the "scan operator" are very similar to those resulting from reduction with the additional attribute that intermediate results are included in the resulting value. That is, the scan derived functions do not change the rank or shape of their argument while the reduction derived functions give results which, in general, are of rank one less than the arguments (hence the name reduction). Note that when a reduction derived function is applied to a scalar argument (rank 0) no action at all is taken on the argument. The result is simply the argument itself. You can observe this attribute by noting the result of an expression such as ∧/9.

The reduction and scan operators have alter egos which are known as compression and expansion. Compression and expansion generate derived "selection" functions whose properties are well known. The interesting characteristic here is that the argument of the monadic operator is a value instead of a function, but the result is, as always, a function.

With two exceptions, all primitive functions and operators are represented by single-character symbols. The exceptions are the indexing function and the axis operator, which are both represented by a pair of matching brackets that enclose their right arguments. The axis operator is dyadic and takes the function (perhaps a derived one) on its left and a scalar value on the right. It generates a derived function which acts on a particular axis of its argument. The use of the axis operator is illustrated by expressions such as *J*⌽[*K*] *A* or +/[*K*] *A* or *A*,[⎕*IO*] *B*.

The above discussion covers all of the operators currently defined in most *APL* systems. Since the major thrust of the continuing development of *APL* involves the addition of new operators as well as new functions it is essential to understand the concepts of "operators" and "functions". For those readers interested in exploring these ideas in much greater depth I recommend the paper **OPERATORS AND FUNCTIONS** by Dr. Kenneth E. Iverson of the APL Design Group, IBM Research Division. The report is available as Research Report number RC 7091 (30339) from the library of the T. J. Watson Research Center, Yorktown Heights, New York 10598.

A Long Boring Task

Stephen Taylor, Copenhagen

The rain was dribbling down the outside of the window in wriggling transparent lines. I lifted my coffee cup, and watched the dull, grey Danish summer through the steam. It didn't look any better.
- "You do it," I said.
Knud didn't look up from his newspaper.
- "I'm busy."
- "But I'm really busy."
- "I'm busier than you are." I looked over at him. He was right; he was doing the crossword.
- "Sixty fields," I protested. "Some with multiple values. And you say yourself that most of them won't be used."
- "You'll think of something."

The problem was an input conversation which had to be written. Each record contained over sixty fields, most of which would remain empty. For this, we had to write a conversation which allowed the user to choose which field he wanted, prompted him for it and checked that its value was permissible. What a way to spend July.

Nearly all the fields were going to be the same. Well, almost: date fields, numeric fields, alpha text, options chosen from a menu; the same few lines of code with tiny variations. Only the prompts and help messages changing. Now wait a minute. Couldn't we make something of that?

Suppose we divided all the fields into five types:

> **Integer** fields are easy.
> **Real** fields can be stored as integers if we know what power of ten to multiply them by.
> **Date** fields can be stored as absolute day numbers.
> **Codes** can be represented as 'vertical indexes' into character menu matrices.
> **Alpha** text is alpha text.

This suggests five corresponding TASK functions: *ΔASKI ΔASKR ΔASKD ΔASKO* and *ΔASK*. If we represent each field as a variable named after its **Field ID**, we can describe what we need in order to prompt for it.

> **Field ID:** a number to identify the field by.
> **Field type:** a number to represent one of the five types above.
> **Field limits.** For integers, the number of digits; for reals, the number of decimal places; for codes, the Messagefile component on which the menu is stored; and for alpha text, the maximum number of characters.
> **Help message.** We can represent this by the Messagefile component number it is stored on, or by a zero if there isn't a help message.
> **Prompt:** a name for the field.

So we could establish two globals *FIELDS* and *FNAMES* which would contain a numeric directory for the fields, and their names. *FIELDS* would correspond to the first four items in the list above, and *FNAMES* to the prompts. With one more global *DATELIMITS* to define the limits within which dates may be selected, we can write a function to get a value for any field.

```
     ∇ R←PROMPT ASKV ID;FIELD;NAME
[1]    I←FIELDS[;1]⍳ID ◇ FIELD←FIELDS[I;] ◇ NAME←FNAMES[I;]
[2]    →L1 IF 0=⎕NC 'PROMPT'
[3]    PROMPT←ΔTR NAME,': ',('◇',⍕FIELD[4]]] IF FIELD[4]≠0
[4]   L1:→(INTEGER,REAL,DATE,CODE,ALPHA)[FIELD[2]]
[5]   INTEGER:R←(1,FIELD[3]]] ΔASKI PROMPT ◇ →ΔEN
[6]   REAL:R←(1,FIELD[3]]] ΔASKI PROMPT ◇ →ΔEN
[7]   DATE:R←ΔDNO DATELIMITS ΔASKD PROMPT ◇ →ΔEN
[8]   CODE:R←FIELD[3] ΔASKO PROMPT ◇ →ΔEN
[9]   ALPHA:→ΔEN I̲FNULL R←ΔASK PROMPT ◇ R←FIELD[3]↑R
[10]   ΔEN:
     ∇
```

Now we had a function that would fetch a suitable value from the terminal for us. However, some of our fields were to have multiple values. We needed a function *GETV* to edit and store new values for fields. We added a fifth column to *FIELDS*, which contained the maximum number of values the field might have. If we decided that all numeric fields were to be vectors, and all character fields matrices, we were ready.

```
    ∇ PROMPT GETV ID;HELP;NVALS;FNAME;FVALUE;NAME;VALUE;IND
[1]    I←FIELDS[;1]⍳ID ◇ HELP←FIELDS[I;3]  ◇ NVALS←FIELDS[I;5]
[2]    FNAME←'F',⍕ID ◇ FVALUE←⍎FNAME
[3]    →GETVALUE IF 0≠⎕NC 'PROMPT'
[4]    PROMPT←ΔTR NAME,': ',('◇',⍕HELP]] IF HELP≠0
[5]   GETVALUE:→END IFNULL VALUE←PROMPT ASKV ID
[6]    →ERASE IF 0≠IND←FVALUE ΔCONT VALUE
[7]    →(SINGLE,MULTIPLE)[1+1<NVALS]
[8]   SINGLE:FVALUE←VALUE ◇ →END
[9]   MULTIPLE:→TOOMANY IF NVALS≤1↑⍴FVALUE
[10]    FVALUE←FVALUE,[1]  VALUE ◇ →END
[11]   TOOMANY:'NO MORE VALUES; YOU MUST ERASE FIRST'
[12]    FVALUE ◇ →GETVALUE
[13]   ERASE:FVALUE←(IND≠⍳1↑⍴FVALUE)⌿FVALUE
[14]    'ERASED' ◇ →GETVALUE
[15]   END:⍎FNAME,'←FVALUE'
    ∇
```

The final function contained extra code to deal with interdependent fields where values had to be entered in parallel, (for example, delivery dates, payment dates, amount invoiced, authorisation) but I have omitted it here for the sake of clarity. With these two functions written, our conversation program became:

```
   ∇ INPUT;ID
[1]  P1:→ΔEN IFNULL ID←FIELDS[;1] ΔASKNO2 'FIELD ID: '
[2]   GETV ID ◇ →P1
[3]  ΔEN:
   ∇
```

Knud looked up from 'Politiken'. "What's work done in a selected field?" he asked. "Four letters." – " TASK "

In the example above, the function *ΔDNO* converts timestamps into absolute day numbers; and *ΔCONT* is a lookup function.
The TASK functions can be found in workspace 999 *TASK*. The functions *ΔASKI* and *ΔASKR* are both extended versions of the function *ΔASKN*, and will shortly be appearing in a new release of the subsystem; please contact the author for details.

*** SPINNING SHELLS Contest Results ***

Congratulations to all who took part in our seventh contest.

The entrants and their submissions were:

Greg Jaxon, Cleveland	*SPIN∆GPJ*
Joey Tuttle, Palo Alto	*SPIN∆JKT*1/2
Nick Telfer, England	*SPIN∆NT*1/2/3/4/5
Roger Hui, Edmonton	*SPIN∆HUI*1/2
Garvin Boyle, Ottawa	*SPIN∆GB*
John Craig, Aberdeen	*SPIN∆JMC*
Ken Pawulski, Toronto	*SPIN∆KPT*
Dr. Joseph Specht, Stiftung Rehabilitation Ctr., Heidelberg	*SPIN∆SPE*1/2
Robin Chesters, Winnipeg	*SPIN∆RCH*
Dinos Appla, Gloucester	*SPIN∆DINO*1/2
Wolfgang E. Fendt, Stiftung Rehabilitation Ctr., Heidelberg	*SPIN∆WEF*1/2
Stuart Baker, Gloucester	*SPIN∆STUB*
Walter Bluger, Ottawa Dept. Nat'l Health & Welfare	*SPIN∆WBL*
Mike Hollosi, Toronto	*SPIN∆MHO*
Mike Powell, Calgary	*SPIN∆POW*
Bill Levis, Victoria, B.C. M'try of Economic Dev't	*SPIN∆WRL*1/2
Bruno Reiter, Germany Coca Cola	*SPIN∆BRU*,
Zeke Hoskin, Vancouver	*SPIN∆ZEKE*1/2
Henri Brudzewsky, Denmark	*SPIN∆HENRI*1/2/3
Keith Morris, Victoria, B.C. M'try of Economic Dev't	*SPIN∆KEI*1/2/3/4/5
Yuen K. Wong, Ottawa	*SPIN∆YKW*

Due to a possible misinterpretation of the specification of the left argument (i.e. the shells were supposed to increase in index from the centre shell out, not vice versa), all but seven submissions were judged to be complete. The seven that were excluded did not satisfy the specifications when either *SF* or ⌽*SF* was supplied as the left argument. The complete solutions were then subjected to time tests with arguments of different sizes.

Most of the solutions were quite fast. The surprising thing, though, is that the five fastest solutions contained a loop. It was necessary to test for workspace efficiency to pick a winner. Again, and not surprisingly, the looping solutions out-performed the others, with the three fastest solutions proving to be the most efficient.

Execution times of these tests expressed as multiples of the fastest entry are as follows:

Entry	Ranked Time
*SPIN∆WEF*2	1.00
*SPIN∆KEI*5	1.06
SPIN∆JMC	1.11
*SPIN∆WEF*1	1.13
*SPIN∆KEI*4	1.31
SPIN∆POW	1.33
*SPIN∆SPE*1	1.40
*SPIN∆HUI*1	1.41
SPIN∆GPJ	1.43
*SPIN∆KEI*3	1.43
*SPIN∆KEI*1	1.46
SPIN∆WBL	1.48
SPIN∆STUB	1.51
*SPIN∆DINO*1	1.57
*SPIN∆NT*3	1.69
*SPIN∆NT*2	1.72
*SPIN∆ZEKE*2	1.79
*SPIN∆ZEKE*1	1.85
SPIN∆RCH	2.23
*SPIN∆YKW*1	2.38
*SPIN∆KEI*2	2.60
*SPIN∆HENRI*2	2.65
*SPIN∆SPE*2	3.26
*SPIN∆WRL*1	3.33
*SPIN∆WRL*2	3.38
*SPIN∆HUI*2	4.84
*SPIN∆DINO*2	7.08
*SPIN∆NT*1	9.41
SPIN∆MHO	21.48
SPIN∆GB	46.31
SPIN∆BRU	56.30

Thus, based on the above analysis, our overall winner is *** **Wolfgang E. Fendt** ***, with his *SPIN∆WEF*2 entry. His code was not only concise, but a pleasure to read. Keith Morris of the B.C. Ministry of Economic Development deserves a special honorable mention, as his *SPIN∆KEI*5 was almost as good as the winner's. The IPSA winner is *** **John Craig** ***. The winners will be contacted shortly, and will receive a book prize of their choice.

Winning entries can be viewed in workspace 999 *CONTEST*. Many thanks to all who participated.

Mike Holloway, Toronto

Paragraphics Perfected

Bob Smith, 6 February, 1979

I have a belated solution to the paragraphics contest which I'd like to pass along.

The function *TXED* in workspace `1729` *TXED* solves the problem with zero (`0`) loops. The condition placed upon solutions that they not loop based upon the number of lines in the resulting paragraph wasn't met (in spirit) by the other solutions. While they might not loop exactly *N* times, sometimes they loop more!

The listing of the function shows how ⌹ plays a key role in the answer. The fundamental technique I used is due to Burton C. Gray who figured out how to solve what is called the transitive closure problem.

The function *TXED2* solves the simpler problem where one assumes no words longer than the width and no multiple blanks:

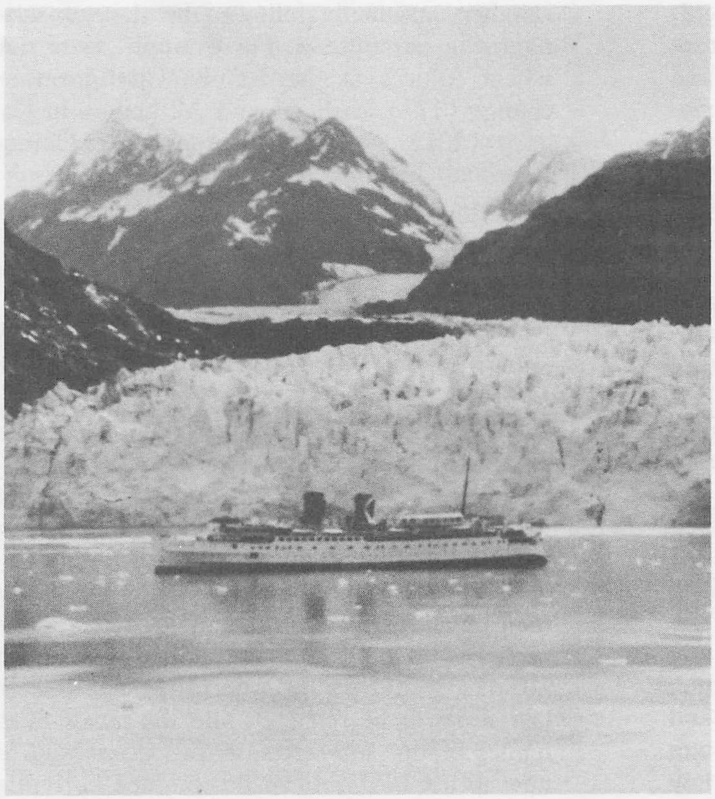
```
     ∇ TXT←W TXED TXT; ⎕IO; A; B; C; E; I; P; RB; LB
 [1]    ⍝ solves text editing problem.
 [2]    ⍝ assumptions:
 [3]    ⍝       ∘ first char is non-blank;
 [4]    ⍝       ∘ all chars are single forward spacing;
 [5]    ⍝       ∘ words longer than W chars broken into W char words;
 [6]    ⍝       ∘ multiple blanks allowed, each line left-justified.
 [7]    ⍝ see TXED2 for clearer solution to simplified problem.
 [8]    ⍝ N.B.: has ws full problems if too many blanks in TXT.
 [9]    ⍝ written by Bob Smith, 4 Feb 79.
[10]    ⍝ revised at suggestion of Leslie Goldsmith, 26 Feb 79.
[11]    ⍝ key algorithm of transitive closure is due to Burton C. Gray.
[12]     ⎕IO←1 ◊ B←TXT=' '
[13]     A←B⍱1↓B,1 ⍝ selects all but last non-blank in a sequence of non-blanks
[14]     P←A/¯1↓1,B ⍝ partition vector on long words for following partitioned +\
[15]     C←P/⍳⍴P ◊ B←B∨A\0=W|+\1-P\C-¯1↓1,C ⍝ mark breakpoints in long words as pseudo-blanks
[16]    ⍝ main portion of algorithm
[17]     RB←B>1↓B,0 ⍝ right-hand blank in sequences of blanks
[18]     LB←B>¯1↓0,B ⍝ left-hand blank in sequences of blanks
[19]    ⍝ for each value in A, the corresponding value in C is the
[20]    ⍝ index in A of the rightmost blank within W+1 characters.
[21]     C←+/((W+1)+0,RB/⍳⍴RB)∘.≥0,(LB/⍳⍴LB),1+⍴LB
[22]     I←(2⍴⍴C)⍴(1+⍴C)↑1 ⍝ ye old identity matrix
[23]     A←(RB\1=1↓((⍴C)↑1)⌹I-(⍳⍴C)∘.=C)/⍳⍴TXT ⍝ the key is transitive closure on C∘.=⍳⍴C
[24]    ⍝ make room for CR's
[25]     E←((⍴TXT)+⍴A)⍴1 ◊ E[A+⍳⍴A]←0 ◊ TXT←E\TXT ◊ TXT[(~E)/⍳⍴TXT]←CR
     ∇

     ∇ TXT←W TXED2 TXT; ⎕IO; A; B; C; E; I
 [1]    ⍝ solves simplified form of text editing problem.
 [2]    ⍝ assumptions:
 [3]    ⍝       ∘ first char is non-blank;
 [4]    ⍝       ∘ all chars are single forward spacing;
 [5]    ⍝       ∘ no words longer than W chars;
 [6]    ⍝       ∘ no multiple blanks.
 [7]    ⍝ see TXED for solution to more general problem.
 [8]    ⍝ N.B.: has ws full problems if too many blanks in TXT.
 [9]    ⍝ written by Bob Smith, 4 Feb 79.
[10]     ⎕IO←1 ◊ B←TXT=' ' ◊ A←0,B/⍳⍴B
[11]     C←+/(A+W+1)∘.≥A,1+⍴B ⍝ for each value in A, C corresponds to the index in A of the
                                  rightmost blank within W+1 characters
[12]     I←(2⍴⍴C)⍴(1+⍴C)↑1 ⍝ ye old identity matrix
[13]     A←(B\1=1↓((⍴C)↑1)⌹I-(⍳⍴C)∘.=C)/⍳⍴TXT ⍝ the key is transitive closure on C∘.=⍳⍴C
[14]    ⍝ make room for new CR's
[15]     E←((⍴TXT)+⍴A)⍴1 ◊ E[A+⍳⍴A]←0 ◊ TXT←E\TXT ◊ TXT[(~E)/⍳⍴TXT]←CR
     ∇
```

ALASKA CRUISE SHIP — A Reservation System

Carol Aitchison, Vancouver

APL programmers can generally reduce most business applications to a simple file system and a few functions, but sometimes ⎕*READ*'s and Dyadic Iotas seem pretty humdrum. We appreciate it when, among the mortgage schedules and standard deviations, some intriguing and even romantic programming projects turn up, such as this one when we were asked to set up a reservation system for the Alaska cruise ship, Princess Patricia.

The Princess Patricia was built on the Clyde, in Scotland, in 1949. Her original service was on a triangle run between Vancouver, Victoria, and Seattle as a passenger and car ferry. In the mid sixties, she was converted to a cruise ship, to replace the Princess Louise (which is now a restaurant in Long Beach, California).



The 'Pat' is operated by the British Columbia Coastal Steamship Service - a division of C.P. Rail. She sails 18 times yearly between May and October, passing some of the most spectacular scenery in the world. Departing from Vancouver, she cruises north through the famed Inside Passage with the first port of call being Ketchikan, in southeastern Alaska. Then, on to Wrangell, through the Wrangell Narrows and overnight into Glacier Bay. A full day is spent observing the glacier.

From there, she proceeds to Skagway where another day is spent. From Skagway she turns southbound and heads to Juneau, the state capital. Then she passes through Tracy Arm, a fjord to rival Norway's best. Next on to Prince Rupert, B.C., Alert Bay, and finally back to Vancouver. The cruise lasts 8 days, and the Pat comfortably accommodates 320 passengers.

In past years, all reservations and inquiries were handled manually by flipping through berthing books, waitlists, and filing cabinets. These often became quite tattered by the time the 5700th passenger had been booked. Although the staff handled the situation capably, the number of repetitive tasks could be cut down by automating, and the amount of ship space sold could be increased. From a programmer's point of view, this was quite a challenge! The system requirements were only loosely defined. The ease of use for the reservation agents was extremely important. The variety of cabin space and tariff arrangements, as well as special cases increased the trickiness of the coding. The result - PPARS - the Princess Patricia Automated Reservation System - was implemented in early January 1979. The following facilities are now available: space inquiries, tariff calculations, reservations, waitlist interface, correspondence, some accounting, ticket printing, and management reports. Two video terminals are used for reservations and inquiries, and one hard copy unit takes care of reports, form letters, and ticket printing.

AGDATA - AGRICULTURAL COMMODITIES DATA BASE

Lloyd Sereda, Alberta Agriculture, Edmonton
Roger Hui, I.P. Sharp Associates, Edmonton

AGDATA consists of time series data that contains mainly price and volume information on a number of key agricultural commodities. The data base was developed as an agricultural supplement to the I.P. Sharp Associates CANSIM* Mini Base, which was extended to include selected data from the Handbook(s) of Agricultural Statistics.

Data to be included in AGDATA were selected on the basis of their interest to agriculture economists and statisticians. Agriculture decision-makers, agribusiness managers, and commodity traders have also exhibited keen interest in the data.

Whereas the data in CANSIM seem best suited for deciding long-term trends, AGDATA series are designed to address more immediate issues, with daily, weekly, and monthly periodicities. Prices, volumes, and indices are gathered from the following sources: Agriculture Canada, Canadian Grain Commission, Canadian Livestock Feed Board, Chicago Board of Trade, Chicago Mercantile Exchange, Grain Elevator Operators, Kansas City Grain Exchange, Minneapolis Grain Exchange, Omaha Cattle Markets, Statistics Canada, United States Department of Agriculture, Winnipeg Commodity Exchange.

A fairly elaborate network has evolved to pull the various series into one data base, and the fact that the data came from many sources seems to make AGDATA useful to a variety of agricultural commodity watchers - say, compare prices of the same commodities at different locations. The acquisition of many additional series is in progress. Each AGDATA series is assigned a series number, currently grouped as follows:

Series	**Commodities**
1 - 200	Grains and Oil Seeds
201 - 300	Cattle and Beef Products
301 - 350	Hogs and Pork Products
351 - 400	Broilers
401 - 450	Eggs and Fowl
451 - 475	Economic Indicators
476 - 600	Special Crops & Miscellaneous

Every AGDATA series is composed of one or more related sub-series, such as *MAR* and *SEP* of the series 177 *WHEAT FUTURES*. Sub-series of a series are referred to by their names.

All retrievals from AGDATA are performed through the function *AGDATA* in conjunction with the functions of MAGIC (in workspace 39 *MAGIC*). To retrieve a particular series, the argument to *AGDATA* should include the series number, optionally followed by the sub-series names in parentheses. For example, to retrieve wheat futures at the Kansas City grain exchange (177), cash A1 and A2 heifers in Calgary (201) and oat futures at the Chicago Board of Trade, type:

```
CLEAR
1 2 3 4 5 DAILY, DATED 26 1 78 TO 4 5 78
PUT AGDATA '177,201(LOW,HIGH),187(JUL)'
```

The time frame is set at daily (weekdays only) from January 26 to May 4, 78 using the MAGIC functions *CLEAR*, *DAILY*, *DATED*, and *TO*. (Monthly or quarterly time frames are also acceptable.) All sub-series of series 177, the sub-series *LOW* and *HIGH* of series 201, and the sub-series *JUL* of series 187 are then retrieved. Finally, the MAGIC function *PUT* is used to save the results of *AGDATA* for further manipulations. If the MAGIC options *AUTOTITLE* and *AUTOLABEL* are in effect, *AGDATA* would set the title to the name of the last series to be retrieved and the labels to the names of the sub-series. A full description on how to use *AGDATA* can be obtained by typing:

```
    )LOAD 39 MAGIC
    AGDATA 'DESCRIBE'
```

AGDATA is managed by the **Economic Services Division of Alberta Agriculture**, Edmonton, Canada. For enrolment details, or enquiries concerning AGDATA, please contact Lloyd Sereda at the Alberta Dept. of Agriculture, 3rd floor, 9718 - 107th St., Edmonton, Canada T5K 2C8, or Roger Hui (I.P. Sharp Associates Edmonton), or your local SHARP APL representative.

Deletions for new release of the CANSIM* Mini Base - April 2, 1979

A redefined CANSIM Mini Base will be available on the SHARP APL system on April 2, 1979. All Mini Base users should receive a copy of the CANSIM Mini Base Review - "Summary of Changes" and an amendment to the CANSIM Mini Base Series Directory 1978. If you have not received these by the end of March, please contact either your local Statistics Canada representative or your local SHARP APL representative.

The following is a list of the matrices which will have some or all series deleted from the CANSIM Mini Base, effective April 2, 1979. If there are any in this list which you would like to retain as part of the Sharp CANSIM Mini Base Supplement, please contact your local SHARP APL representative before March 27,1979.

Matrix	Subject
31	Production of rigid insulation board
59	New motor vehicle production
63	Newly completed unoccupied dwellings
67	Sales-made in Canada sets, radio receivers + televisions
68	Electric power statistics
88	Raw hide, skins + finished leather
91	Producers' shipments of non-metallic minerals
122	Domestic shipments of building matrials
131	Wholesale price index - 30 industrial materials
140	Sugar statistics
179-180	Consumer credit
184	Manufactured food
185	Res. + non-residential building construction input price index
293	General wholesale price index
428-430	Consumer price index
671	Industry selling price index - petroleum + coal
807-808	Wages and salaries
917	Chartered banks - general loans
2064-2065	Labour force survey - Canada, Nfld. - annual
2073,2076	Labour force survey - Alta., B.C. - annual
2554	Export price indices
2781-2839	Industrial corporations - financial statistics (replaced by matrices 6781-6845)
2877-2886	Farm input price indexes
3094,3116,3128	Quarterly estimates of job vacancies, Canada, Nfld., N.S.
3134,3140,3163	Quarterly estimates of job vacancies, Que., Ont., Man.
3209,3215,3244	Quarterly estimates of job vacancies, Sask., Alta., B.C.
3256	Job vacancy rates - Canada and provinces
7100-7113	Consumer price index - Canada
7127	Consumer price index - Vancouver (replaced by mat. 7000-7031 Canada and regions)

* CANSIM is the registered Trade Mark for Statistics Canada's machine-readable data base. When publishing any data retrieved from CANSIM, the following must be used as the source: "These data originate from CANSIM which is the registered Trade Mark for Statistics Canada's machine-readable data base."

NEW EQUIPMENT FOR THE COMPUTER CENTRE

Bill Apsit, Toronto

Early in March we are installing a second Amdahl 470/V6-II computer. It arrives with 4 million bytes of main memory. After we verify that it operates reliably, we will increase the memory of our original Amdahl from 3 to 4 million bytes. The new computer replaces our remaining IBM 360/75. Unless we find a new home for the 360/75, it will reside in storage.

This installation will provide us with a twin for backup of our current Amdahl, and a more powerful 'slave' CPU for peak-load times. The additional storage enables us to increase the sytem workspace size again - which we will do. At off-peak times the additional system will provide us with an ideal test and development machine. This is particularly valuable to us for our production of 'in-house' SHARP APL systems.

Prior to this new computer installation, we have built and switched to a significantly improved electrical power environment. It provides us with additional power for the new Amdahl, for a new air-conditioner later this year, and for further growth. We do not expect to reach the limits of power now available. It includes a built-in spare transformer, very nice should we ever require it. Connecting our new electrical equipment resulted in reduced hours of SHARP APL availability on some past weekends. This is now complete, and our new centralized, isolated and capacious power system should serve us well.

RELIABILITY FIGURES FOR 1978

Period	Sched hrs	Crashes	Mins lost	% up
October	546	8	120	99.6
November	548	10	154	99.5
December	542	1	14	99.9
4th Quarter 78	1636	19	288	99.7
3rd Quarter 78	1621	27	275	99.7
2nd Quarter 78	1612	24	346	99.6
1st Quarter 78	1603	24	346	99.6

APPLICATIONS LIBRARY UPDATE

Library Workspaces

NEW	35 *PRINCIPAL*, see page 7
	123 *GRADUATION*, see page 7.
CHANGED	39 *X11*, see page 7
	39 *MAGIC*, *AGDATA* function added, see page 10.
REMOVED	34 *PRINCIPAL*, replaced by 35 *PRINCIPAL*.

ACTUARIAL LIBRARY UPDATE

The contents of components 91 and 92 of file '123 *MORTABS*' have been modified. These components contain the C7072 (male and female) life tables as per the statistics Canada catalogue publication number 84-532. This information was previously stored as vectors of *LX* values. It now resides in the file as vectors of *QX* values. The reasons for this change are twofold:

(a) The *LX* values which were present could not be used to accurately reproduce the published values of *QX*. Now however, with the published values of *QX* in the file, the published values of *LX* may be reproduced with reasonable accuracy via code of the following form:

```
LX←⌊0.5+×\100000,1-¯1↓QX
```

(b) The Provincial Life tables from the same study (i.e. components 98 to 115 inclusive) are all stored as *QX* vectors. Components 91 and 92 are therefore now more consistent with their provincial counterparts.

We regret any inconvenience that this change may bring about, but advise that users who are accessing this information via the functions in workspace 123 *ACT* will be unaffected. Lastly, holders of the new ACTPAK user documentation are advised to pencil in the appropriate modifications to the Appendix of their manual.

APL COURSES

INTERMEDIATE:
Gloucester, April 2, August 13
London, March 27-28, May 3-4, June 7-8, July 3-4,
Los Angeles, April 24-26
Montreal, May 14-16, July 16-18
San Francisco, April 4, June 6, October 9, December 4
Seattle, March 6-8
Stockholm, (in Swedish or English)
Toronto, March 19-21, May 22-24
Washington, April 20

ADVANCED:
Birmingham, 'APL and System Design', May 14
London, 'APL and System Design', March 29-30, April 29, May 24, July 19, September 3
San Francisco, August 7
Toronto, 'Advanced APL & Efficient Coding Techniques', April 5

SPECIAL COURSES:
'Appreciation of APL'
  Gloucester, March 5, May 7, July 2, September 3
  London, March 1, April 26, June 4, July 31, September 20
  Warrington, May 16, September 19
'APL Review'
  London, March 26, May 2, June 6, July 2, August 8, September 24

'Aviation Data Base'
  Los Angeles, March 20-22
'Use a terminal'
  London, May 11

SEMINARS:

Birmingham	Advanced Operators (1/2 day), March 30
	Formatting with SHARP APL (1/2 day), April 27
	Restart, May 25
	Highspeed Print, June 29
Edmonton	Statistical Analysis, March 15
Gloucester	Actuarial, August 20
	Advanced Operators, April 30
	APL Idioms, May 14
	Box-Jenkins, June 6
	Files , June 25
	Financial , July 16
	Forecasting, March 26
	Graphics/Plotting, April 23
	Linear Programming , September 17
	MABRA, March 19
	MABRAUTIL, July 9
	Project Planning, May 21
	Regression, Apr 17
	Report Formatting, September 10
	Security, September 24
	Word Processing, March 12
London	Efficient File Design, April 5
	N- and B-tasks, September 6
	Plotting with Magic/3 Plot/Grafplot, June 28
	Report Formatting, March 6
	Sharp Network Facilities, April 24
	Your Bugs - How to get rid of them , May 22
Los Angeles	Data Base Design, May 15
Montreal	Financial Analysis, July 5-6
	PERT, May 2
	Plotting with SHARP APL, June 12
	Saving Money with N- and B-tasks, April 10
San Francisco	Graphics, March 14, May 9
	MAGIC, June 20, September 19
Seattle	Forecasting with SHARP APL (1 day), March 21
Toronto	Actuarial APL Techniques (1 day), April 9, June 8
	AIDS - Introduction (2 days), April 10-11, June 14-15
	Box-Jenkins (1 day), March 5, May 8
	Data Base Design (1/2 day), April 26, June 22
	Forecasting Methods (1 day), March 7, May 11
	Graphics (1 day), April 6, June 11
	Magic for Time Series Analysis (1 day), March 9, May 7
	Plotting (1 day), March 16, May 10
	Regression Analysis (1 day), April 18, June 13
	Report Formatting (1/2 day), April 20, June 20
	Saving Money with N- and B-tasks, April 17, June 4
	Text Editing (3 day), March 26-28, May 28-30

INTRODUCTION TO APL:

Atlanta
(5 day)
May 7-11

Amsterdam
(3 day)
March 21-23

Birmingham
March 12-14
May 8-10

Calgary
(5 day)
April 3,4,
11,18,25

Gloucester
(3 day)
February 12-14
June 4-6
October 7-9

London
(3 day)
March 7-9
April 9-11
May 16-18
June 18-20
July 25-27
September 10-12

Los Angeles
(3 day)
April 10-12

Montreal
French/English
March 7-9/12-14
April 2-4/17-19
May 7-9/22-24
June 4-6/19-21
July 9-11/24-26

Ottawa
March 5-9
April 2-6

Rochester
March 19-23
April 16-20

San Francisco
(4 halfdays)
March 6-9
April 17-20
May 22-25
June 10-13
August 21-24
October 2-5

Seattle
(3 day)
April 10-12

Toronto
(3 day)
March 12-14
April 2-4
April 23-25
May 14-16
June 5-7
June 25-27

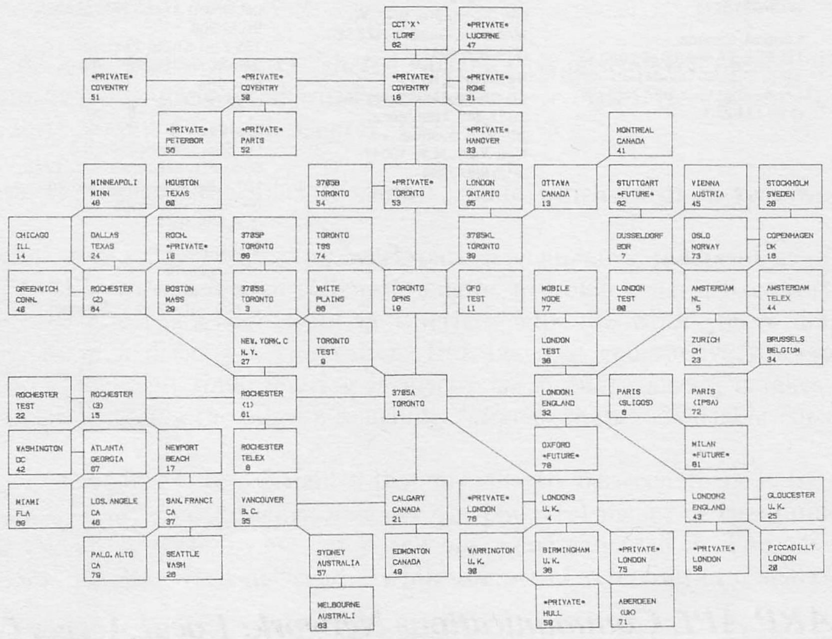
Vancouver
March 21-23
April 25-29
May 23-25

Victoria
(3 day)
April 3-5

Warrinton
March 20-22
June 12-14

Washington
(4 days)
March 26-29
May 21-24

Network Topology



UPDATE

☐ Please amend my mailing address as indicated.	Name: ____________
☐ Add to your mailing list the following name(s).	Co.: ____________
☐ Send me a SHARP APL publications order form.	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 Associates, 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 Associates, 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 Associates 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 St.
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
Corso Plebisciti 15
20129 Milan
Italy
(2) 733 563

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 Center 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

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 – Cêdex 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

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 Associates 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 • Oslo • Raleigh • Regina • Saskatoon • 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*