the I.P. Sharp newsletter

MAY-JUNE 1976

I.P. SHARP ASSOCIATES IN *Europe*

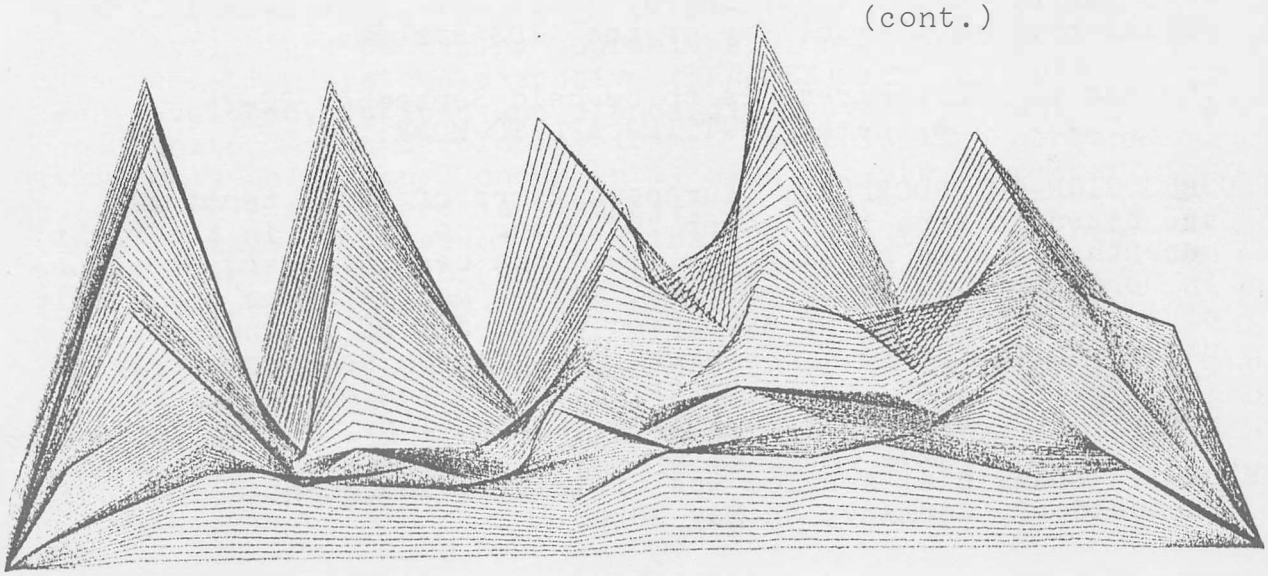
by G.H. Robinson

The third anniversary of the availability of *SHARP APL* in London via a transatlantic cable has just passed and now is an appropriate time to review the European scene from the Sharp viewpoint.

The emphasis to date in our European operations has been centred in the United Kingdom, where local access is now additionally available in Gloucester and will shortly be also available in Warrington (near Manchester) and Coventry. In the UK we are collaborating with the Post Office in their Experimental Packet Switched Service (EPSS) which, according to present plans, will make access to *SHARP APL* available at the year end from anywhere within Scotland, Wales and England at the cost of a local call.

In 1975 the link to London was extended to Amsterdam where customers are serviced by our Dutch subsidiary, Intersystems B.V. Since January 1974, *SHARP APL* has been available in Paris from our colleagues SLIGOS S.A. from their dedicated 360/50. Just recently, after many delays outside our control, a London-Paris link has been commissioned, giving access to those customers in France who may want to access data bases which are common to their operations in North America and France. By the end of June, a direct Paris-Toronto circuit will be commissioned, with the London-Paris link being retained.

(cont.)



SHARP IN EUROPE (cont.)

For the remainder of 1976 and 1977, the major emphasis will shift to the continent of Europe, with a planned, rapid extension of our network. A German subsidiary, I.P. Sharp GMBH, has been established on the banks of the Rhine in Duesseldorf and the long-awaited Amsterdam/Duesseldorf link is now operational. The shape of demand from existing and potential customers is now sufficiently clear for our network planning to include provision for local access in Copenhagen, Stockholm, Hanover, Zurich, Vienna, Brussels, Milan and Rome by the end of 1977. The schedule is, of course, subject to extraneous influences over which again we will have no control.

Already the unique combination of *SHARP APL* and the Sharp transatlantic network has lead to growing use by our existing multi-national customer base and our expansion plans will undoubtedly attract the attention of many other multi-nationals, particularly where their interests lie in the fields of financial reporting, planning and control.

In summary, then, the past three years have seen the secure establishment of *SHARP APL* in the United Kingdom, as a base for expansion to the continent, and also the beginnings of our continental operation. The next three years will, without doubt, produce their problems, but so did the previous ones. *SHARP APL* gained immediate acceptability in the UK, which now makes a major contribution to the corporate time-sharing revenue and profitability. We are hopeful of the same success on the continent.

TAKE NOTE

AIDS A new version of the financial planning package was released this month. Significant improvements have been made in the efficiency and flexibility of the AIDS system.

APL76 The annual APL congress is to be held September 22-24 in Ottawa. The theme: PUTTING APL TO WORK.

ASCII USERS SIGN-ON PROCEDURE: European users of ASCII terminals are asked to type the letter "O" prior to typing in the right parenthesis when signing on. This has been standard procedure in North America for some time, and is necessary as the result of our introduction of autospeed ports and of our network expansion in Europe. BCD users may sign on as usual. Please contact your local representative should you have any difficulties or like to have a fuller explanation.

DUESSELDORF: Local dial service was officially inaugurated in Duesseldorf, Germany, on May 20th.

HIGH VOLUME & HIGH SPEED TRANSMISSIONS

*TO AND FROM THE UK
ROCHESTER N.Y.
CALGARY ALTA.
OTTAWA

by David Booth
& Steve Clarke

For some time now the London, England office of I.P. Sharp Associates has been using an MDS 2400 tape unit and controller to transmit data between London and Toronto. This is data which the customer would not normally wish to transmit through the low speed terminal. For example, there are large files of data (and updates) being copied from customer processors to Sharp's *APL* system so that they can be processed interactively in real time via *SHARP APL*. There are large print-outs being transmitted to the UK from customers' *APL* files. In some cases these are passed on to the customer on magnetic tape for local printing by his own DP department and in others, Sharp London prints the data. In addition to the Toronto-London link, the London MDS 2400 can be linked to a wide range of remote job-entry terminals via the public switched network so that volume data may well be transmitted right to the customer's premises.

There are typically two types of transatlantic transmissions between the UK and the computer centre in Toronto. Usually, tapes transmitted from Toronto to the UK are fileprints originating from the *PREQ* or *AUTOPREQ* or *HSPRINT* functions. Tapes transmitted from the UK to Toronto usually contain bulk data for mounting to file. There are three functions which handle the management of and administration relating to tape transmissions. These functions can be found in the workspace 1099010 *UKMOHAWK* and return a reference number which should be used in enquiries.

Toronto-London tapes are normally transmitted and received in the evening (UK time). If the UK office is notified before noon and the tape received before 5:00 PM (London time) it will be transmitted that evening and should be available for use the next morning (allow longer for unusual formats).

A customer will normally receive a tape transmitted from *APL* file the morning after the day requested. If an *APL* file printout is required then the turn around will usually be 24 hours. Should you wish to transmit data via the MDS 2400, or wish to request volume printouts, ask your local Sharp representative for information. UK users call Steve Clarke on 01-629 1564.

The Toronto-London link is part of Sharp's growing network for the transmission of high volume data which currently includes Toronto, London, Rochester N.Y., Ottawa and Calgary.

*For specific details on the use of your local link contact your *SHARP APL* representative.

PACKET SWITCHING

In earlier newsletters we told you about our plans for a private packet switching network. This is in the form of a progress report to let you know how things are coming along. We are several months behind where we expected to be, but recently considerable progress has been made and the first six nodes of the system are currently operational. These nodes are in all European cities and in Rochester NY.

The mini computer acts as an assembly point for local incoming low speed data. A number of characters (from the same low speed line) are grouped together in a packet together with administrative information, and the packet is output on the high speed synchronous line. The destination is another node in another city and for this purpose the 3705 acts as any other node. Thus packets may pass through several nodes before arriving in the computer room, but at no time are they de-multiplexed before reaching the 3705.

In the event of an error in transmission, the offending packet is merely re-transmitted. The system uses statistical multiplexing, i.e. it takes advantage of the fact that statistically, not all terminals are operating at full speed at the same time. Thus the band width of the communication line may be used more effectively. In an early test between London and Toronto, 12 300 baud terminals were serviced with ease on a 2400 baud channel.

The length of a packet is relatively short, compared with other systems, but this is deliberately done to maintain the fast response time that most APL users have got used to. The remainder of our network will be converted to the new system during the course of the next year or so.

SUMMARY OF DATA BASES AVAILABLE IN *SHARP APL*

1. NBER - U.S. Economy (National Bureau of Economic Research).
   Approximately 2,400 monthly, quarterly and yearly time series concerning the economy of the U.S. updated monthly from NBER in NY.
   Directory available to all via `)LOAD 582 NBUSAGE`
   `DIRECTORY`
   and then use *PRINTREQ* in 1 *FILEPRINT*.
   Access routines: 582 *NBUSAGE* and 702 *JUSTMAGIC*.

2. CANSIM MINIBASE - Canadian Economy.
   Approximately 21,000 monthly, yearly and quarterly time series - see following article.

3. OECD - 25 countries of the Organization of Economic Co-operation and Development. Updated via OECD in Paris.
   Three data bases: Major Indicators (monthly, quarterly & yearly)
   National Accounts (quarterly)
   National Accounts (yearly).
   Directory available from Sharp: 'OECD Data Base Directory'
   Access routines: 582 *OECDUSAGE* and 702 *JUSTMAGIC*.

DATA BASES IN *SHARP APL* (cont.)

4. CAB - Aviation data base for U.S. carriers.
   Three data bases: Form 41 Schedules, as submitted by all major U.S. commercial carriers to the Civil Aeronautics Board (monthly, quarterly & yearly
   Statistics Canada data for 7 Canadian carriers (quarterly)
   Origin-Destination time series - itineraries - for all major U.S. carriers.
   Updated from CAB, Washington D.C.
   Directory available from Sharp: 'CAB/*MAGIC* Directory'.
   Access routines: 702 *MAGIC* and 702 *JUSTCAB*.

5. FINANCIAL POST Security Information - Stock Market, (U.S. & Canada) and Fundamental Facts (Canada).
   Three data bases: U.S. (daily, weekly & monthly)
   Canadian (daily, weekly & monthly)
   Fundamental Facts - balance sheet information for approximately 400 Canadian companies.
   Updated daily from the Financial Post in Toronto.
   Directory available from Sharp: '*SHARP APL* Financial Post Data Bank'.
   Access routines: Public Library 51.

6. NFS - Canadian Economy Forecasts by Informetrica.
   Approximately 2,500 time series containing the results of a large forecasting model written by the Economic Council of Canada. (The model itself does not run on our machine).
   Directory available from Informetrica Limited, Ottawa.
   Access routines: 582 *NFS* and 702 *JUSTMAGIC*.

7. NEELS - National Emergency Equipment Locator System. A data base maintained by Environment Canada for the location of equipment needed for any emergency (e.g. an oil spill) within Canada.
   Directory available from Sharp - brochure from Environment Canada.
   Access routines: 950 *SPILL*.

8. CENSUS71 - 1971 Canadian Census data.
   Directory: *DESCRIBE* in 83 *CENSUS*71.
   Access routines: 83 *CENSUS*71.

9. BANKS - Chartered Banks of Canada.
   42 facts for each of the 10 Chartered Banks in Canada (monthly assets & liabilities)
   Revenue and Expense (quarterly).
   Directory: *DESCRIBE* in 54 *BANKS* and 54 *BANKSQ* .
   Access routines: 54 *BANKS* (monthly) and 54 *BANKSQ* (quarterly).

10. Actuarial Tables.
    Several Mortality tables that may be used in conjunction with *SHARP APL* Actuarial Package.
    Directory available from Sharp: '*SHARP APL* Actuarial Package'
    Access routines: Public library 52.

CANSIM MINI BASE

by Ralph Morrison

On June 1, 1976, I.P. Sharp Associates Limited became the official secondary distributor of Statistics Canada's *CANSIM* mini base. The mini base is a standard subset of approximately 21,000 time series which have been selected as the most frequently accessed and useful of the series in Statistics Canada's main *CANSIM* data base, now numbering over 110,000 time series. While the content of the mini base is only a slight extension of the series previously carried on the *SHARP APL* system, the new data base will be much more timely since it is updated daily from the main base. Access to time series in the mini base has remained unchanged (through access functions in the workspace 81 *CSUSAGE* or 582 *JUSTMAGIC*) so the impact of these new arrangements should be minimal. However, users might note the following facts.

A new series directory of only those series in the mini base has been published and is available from Statistics Canada, in either English or French, for $10.00. This new directory should relieve the uncertainty of whether or not a series found in the Series Directory is carried on the Sharp system. For the convenience of users wishing to order a mini base series directory, an order form may be printed at your terminal by loading workspace 81 *CSUSAGE* and typing: *ORDERFORM*. Send to:

CANSIM,
Current Economic Analysis Division,
Statistics Canada,
23rd Floor, R.H. Coates Building,
Tunney's Pasture,
Ottawa, Ontario K1A 0Z8.

Statistics Canada personnel across Canada will be available to answer questions on content, timeliness and availability of data on the *CANSIM* mini base. In addition they may participate in seminars on Sharp's *CANSIM* offerings and be consulted on questions of expanding the content of the mini base.

As before, *CANSIM* time series that are not carried on the *SHARP APL* system, but which are required by the user, may be obtained from the main base. However, their existence would not be reflected in the mini base directory. An enquiry function, *SOURCE X* (*X* a scalar or a vector of series numbers) is supplied in both of the above workspaces to enable users to determine whether the series is from the *CANSIM* mini base or is an "extra" series.

We hope that these new arrangements will make statistical analysis using socio-economic time series even more convenient and useful, and we welcome comments from users on this change.

ENHANCEMENTS TO THE FILE SUBSYSTEM

by Leslie Goldsmith

Two important modifications have recently been made to the *SHARP APL* file subsystem.

FILE FULL: Previously, the error report *FILE FULL* could occur if the space required for a ⎕*APPEND* or a ⎕*REPLACE* operation exceeded the amount of storage remaining in the file. This meant that programs which performed automatic resizing of files to ensure adequate space for new data had to make rather complex calculations to determine how much space was required. Further, these calculations were often dependent on the physical record length (*PHRECL*) of the file subsystem; this dependence was undesirable as the value of *PHRECL* is not static.

With the recent modification, ⎕*APPEND* or ⎕*REPLACE* operations will be successful (file access permitting) if there is any space remaining in the file, even if it is not sufficient to hold the new data. This makes it legitimately possible to have a file in which the space used exceeds the space allocated, by the additional storage required for the last ⎕*APPEND* or ⎕*REPLACE* operation. However, the next operation to that file requiring more space will produce a *FILE FULL* error report, as before.

This modification greatly facilitates the task of automatically resizing files. For example, prior to the change, a file append program designed to allocate more space to a file if necessary might have looked as follows:

```
    ∇ DATA APPEND FNO;LIM;SIZE;TYPE;⎕IO
[1]   ⎕IO←1 ◇ LIM←⎕SIZE FNO
[2]   TYPE←5↑,DATA ◇ TYPE←4 ⎕WS 'TYPE'
[3]   SIZE←LIM[3]+(PHRECL×6)+(⌈1.2×(TYPE= 20 24 36 56)/ 0.125 1 4
      8)××/⍴DATA
[4]   →(LIM[4]>SIZE)⍴APP ◇ SIZE ⎕RESIZE FNO
[5]  APP:DATA ⎕APPEND FNO
    ∇
```

With the new definition of *FILE FULL*, this can be written much more concisely:

```
    ∇ DATA APPEND FNO;LIM;⎕IO
[1]   ⎕IO←1 ◇ LIM←⎕SIZE FNO
[2]   →(<⌿LIM[3 4])⍴APP ◇ (LIM[3]+5000) ⎕RESIZE FNO
[3]  APP:DATA ⎕APPEND FNO
    ∇
```

The number 5000 is largely arbitrary, and can in fact be made as small as 1 .

No changes are made for allocated file space - only file space actually used.

MEMORY BUFFERS: Previously, a □*APPEND* or □*REPLACE* to a file frequently placed the new data in a temporary buffer, rather than immediately updating the user's file on disk. The temporary buffer was marked as being a "must-write" (*MW*) buffer, and the file subsystem ensured that the data was eventually placed on disk.

The advantages of this procedure were evident mainly for a series of small file updates, which often had only to update an internal buffer, rather than perform a disk operation. However, with large or even medium-sized components, the advantages of *MW* buffers was greatly reduced. More important, data resident in *MW* buffers waiting to be updated on disk at the time of a system failure was inevitably lost.

As the order in which *MW* buffers were actually transferred to disk could differ considerably from the order in which the file operations were originally performed by the user, it was often extremely difficult to know exactly the state of a file after a system failure. This made providing automatic recovery mechanisms which ensured a high level of data integrity a formidable task.

Currently, the file subsystem avoids the temporary placement of data in *MW* buffers, and instead performs the actual disk update immediately. This means that file operations are performed in the order in which they were executed; a new file operation will not begin until a previous one has safely completed on disk. In addition to making it far simpler to design automatic restart mechanisms with the aid of □*LX*, not using *MW* buffers has the added advantage of significantly reducing the CPU time required for file operations.

The changes to file-full considerations and to the internal use of *MW* buffers are both user-transparent, and do not affect the operation of programs predating them. However, users might wish to take advantage of these enhancements in future packages. File subsystem must-write buffers are described more fully in *SATN*-16. The use and maintenance of files is documented in the publication "*SHARP APL* File Subsystem Instruction Manual" (Copyright 1976). Both of these may be obtained from your local *SHARP APL* representative.

THORN The report formatting *APL* primitive (⍕) is described in *SATN*-17.

USAGE INQUIRY SYSTEM: The resources used by a user can be determined by using workspace 1 *USAGE* - see *SATN*-9.

Technical Supplement

WE WOULD LIKE TO THANK ALL OF OUR READERS WHO HAVE PASSED ALONG TO US THEIR COMMENTS AND SUGGESTIONS CONCERNING THE TECHNICAL SUPPLEMENT.

SOME NOTED THAT THE SUPPLEMENT WAS GETTING TO BE TOO LENGTHY, WHILE OTHERS FOUND IT TO BE TOO TRIVIAL.

TO SATISFY BOTH SIDES WE HAVE DECIDED TO RELEASE THE REALLY TECHNICAL AND/OR LENGTHY PAPERS AS SATNS, AND USE THE TECHNICAL SUPPLEMENT AS AN OVERVIEW OF THE USE OF APL IN AREAS OF GENERAL INTEREST, WITH EMPHASIS ON BREVITY AND APPLICABILITY.

IN THIS ISSUE WE ARE PRESENTING A DIGEST OF A SOON TO BE RELEASED SATN ON THE USES OF THE SCAN OPERATOR.

A GRAB-BAG OF SCANS

BY CLEMENT KENT

ALTHOUGH SCANS HAVE BEEN AVAILABLE IN SHARP APL FOR SOME TIME, MANY USERS DO NOT YET SEEM TO HAVE A FIRM GRASP OF THE USE OF THIS IMPORTANT EXTENSION OF THE LANGUAGE. THE FOLLOWING IS A SURVEY OF THE MORE IMPORTANT OR INTERESTING USES OF SCANS.

DEFINITION OF SCAN

MANY PEOPLE ARE A BIT FUZZY ON PRECISELY WHAT IT IS THAT SCANS DO. GIVEN A VECTOR V AND A SCALAR OPERATOR ⍟ SUCH THAT ⍟/V EXISTS,

```
(1)    R←⍟\V ↔ (R[I]=⍟/I↑V)    NOTE ORIGIN 1  (FOR ALL I∈⍳⍴V)
```

THIS DEFINITION SEEMS STRAIGHTFORWARD ENOUGH, AND IT IS WHEN ⍟ IS ASSOCIATIVE. NOTICE THAT IN THIS CASE WE HAVE THE NICE RELATION

```
(2)    R[I+1] ↔ R[I]⍟V[I+1]
```

THE ASSOCIATIVE OPERATORS ARE: ∧ ∨ = ≠ + × ⌈ ⌊. NOTE THAT THE FIRST FOUR OPERATORS ARE ASSOCIATIVE ONLY WHEN V IS A BOOLEAN VECTOR.

MINUS AND PLUS SCAN (-\ AND +\)

NOW, WHAT HAPPENS WHEN ⍟ IS NOT ASSOCIATIVE? LET'S TAKE THE SIMPLEST CASE, WHEN ⍟ ↔ -.

```
      -\⍳5
1 ¯1  2 ¯2  3
```

AS EXPECTED, THE SECOND MEMBER OF THE RESULT IS 1-2. HOWEVER, THE THIRD MEMBER IS NOT (1-2)-3 (=¯4), BUT RATHER 1-(2-3) BECAUSE -\1 2 3 IS CALCULATED BY PERFORMING THE RIGHMOST SUBTRACTION FIRST TO GET 2-3, AND THEN SUBTRACTING THE RESULT FROM 1. THE PECULIARITY OF NON-ASSOCIATIVE SCANS THUS DERIVES FROM THE RIGHT-TO-LEFT EXECUTION RULE OF APL.

'AND' AND 'OR' SCANS (∧\ AND ∨\).

BY DE MORGAN'S LAWS THE TWO ARE RELATED BY ~∧\V ↔ ∨\~V

EXAMPLE:

STOCK[P] IS THE AMOUNT OF STOCK OF PRODUCT P.
DEMAND[P;D] GIVES THE DEMAND FOR PRODUCT P ON DAY D.
1⍴DEMAND ↔ ⍴STOCK.

```
     DAILYSTOCK←+\STOCK,-DEMAND

     DAYOUT←+/∧\0<DAILYSTOCK
```

DAILYSTOCK[P;D+1] GIVES THE STOCK OF PRODUCT P ON HAND ON DAY D.
DAYOUT[P] IS THE DAY THE STOCK OF PRODUCT P RUNS OUT.

TIMES SCAN (×\)

IF V IS BOOLEAN, THEN 1=×\V ↔ ∧\V. NOTE THAT ×\N⍴X IS A FAST WAY OF GETTING X⍳N.*

EXAMPLE:

```
    ∇ Y←CASH PVAL DISCOUNTS
[1]  ⍝ CASH[I] IS THE CASH FLOW IN YEAR I (ORIGIN 1)
[2]  ⍝ DISCOUNTS[I] IS THE COST OF MONEY, INFLATION RATE, ETC
[3]  ⍝  IN YEAR I
[4]  ⍝ Y IS THE NET PRESENT VALUE OF THE CASH FLOWS.
[5]   Y←+/CASH÷×\(⍴CASH)⍴1+DISCOUNTS÷100
     ∇

      100 200 250 PVAL 10 11 12
437.5219375219376
```

NOT EQUAL AND EQUAL SCAN (≠\ AND =\)

THE TWO EXPRESSIONS ARE AGAIN RELATED BY =\~V ↔ ~≠\V

SUPPOSE WE HAVE A STRING V OF ZEROES AND ONES IN WHICH THE ONES MARKED OFF THE BEGINNINGS AND ENDS OF SUBSTRINGS. HOW CAN WE GET A VECTOR R CONTAINING A 1 FOR EVERY MEMBER OF THE SUBSTRINGS?

```
      R←V∨≠\V
```

WE CAN THINK OF THE ≠\ AS REPRESENTING A 'FINITE-STATE MACHINE' WHICH IGNORES 0'S (0≠X IS THE SAME AS X) AND CHANGES STATE WHEN IT ENCOUNTERS A 1 (1≠X ↔ ~X).

SO, STARTING WITH A 0 SIGNIFYING MAIN STRING, WE GET 0'S IN THE SCAN UNTIL WE ENCOUNTER THE 1 AT THE BEGINNING OF A SUBSTRING, AT WHICH POINT THE RESULT OF THE SCAN BECOMES 1 IT REMAINS 1 UNTIL WE ENCOUNTER THE 1 MARKING THE END OF THE SUBSTRING AT WHICH POINT THE RESULT BECOMES 0 AGAIN.

EXAMPLE:

```
      V←14⍴ 0 0 1 ◇ V,[.5] ≠\V
0 0 1 0 0 1 0 0 1 0 0 1 0 0
0 0 1 1 1 0 0 0 1 1 1 0 0 0
```

EXAMPLE:

```
      TEXT←'MAIN TEXT∘SUBSTRING∘MORE TEXT∘ANOTHER SUBSTRING∘STILL TEXT'

      V←'∘'=TEXT ◇ R←≠\V

      R/TEXT
∘SUBSTRING∘ANOTHER SUBSTRING

     (~R)/TEXT
MAIN TEXT∘MORE TEXT∘STILL TEXT
```

EXAMPLE:

```
      OUTPUT←(+/∧\TXT=' ')⌽TXT
```

TXT IS A CHARACTER MATRIX NOT NECESSARILY LEFT ALIGNED. THE OUTPUT IS LEFT-JUSTIFIED.

MIN AND MAX SCANS (⌊\ AND ⌈\)

EXAMPLE:

A BANK DECIDES TO OFFER A SAVINGS ACCOUNT FOR WHICH INTEREST IS CALCULATED DAILY ON THE MINIMUM BALANCE TO DATE AND COMPOUNDED MONTHLY. IF MI IS THE MONTHLY INTEREST RATE, DAYS THE NUMBER OF DAYS IN THE MONTH, AND BALANCE[A;D] IS THE BALANCE IN ACCOUNT A ON DAY D, THEN

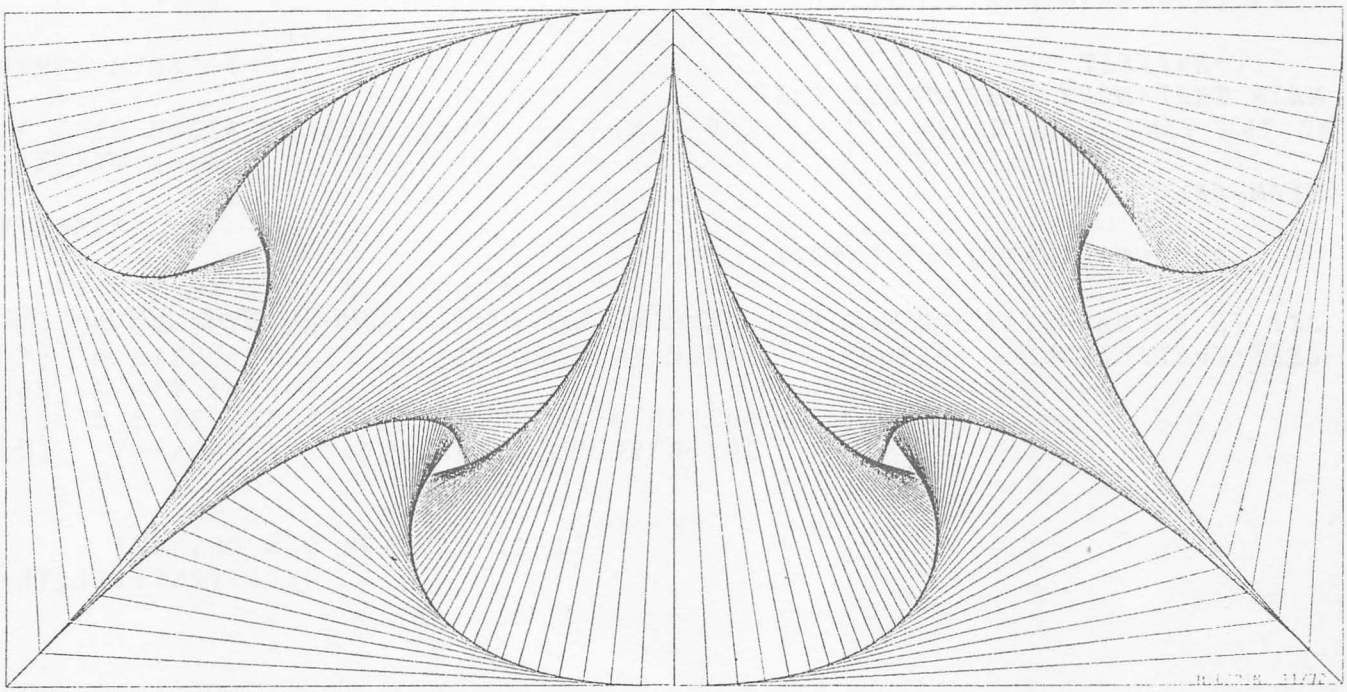
```
INTEREST←+/(⌊\BALANCES)×(1+MI÷100)*÷DAYS
```

SUMMARY

ALTHOUGH SCANS DO NOTHING THAT COULD NOT BE DONE WITH PROGRAMMED LOOP OR RECURSIVE FUNCTIONS, IN MANY CASES THEY ARE BOTH SIMPLER TO USE AND MUCH MORE EFFICIENT. ALTHOUGH IT MAY AT FIRST SEEM DIFFICULT TO PROGRAM USING SCANS, THE BASIC COMPONENTS ARE SIMPLE TO LEARN. A LITTLE STUDY CAN PAY OFF HANDSOMELY.

QUESTIONS, COMMENTS, AND CONTRIBUTIONS ARE WELCOME AND SHOULD BE ADDRESSED TO:

HAL CARIM (MAILBOX 'HCA')
I.P. SHARP ASSOCIATES,
SUITE 1400,
145 KING STREET WEST,
TORONTO,
ONTARIO M5H-1J8



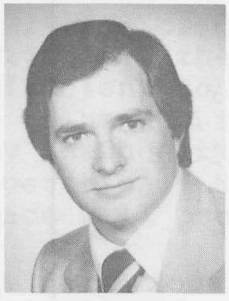
SHARP APL COURSES

INTRODUCTION TO *APL*	June	July	August
DALLAS	21-25		
LOS ANGELES	7-11	5-9	2-6
OTTAWA	7-11	5-9	2-6
ROCHESTER	21-25	19-23	23-26
TORONTO	14-17	12-15	9-11
UK	2-4,7,8		
INTERMEDIATE *APL*			
TORONTO	28-30		23-25

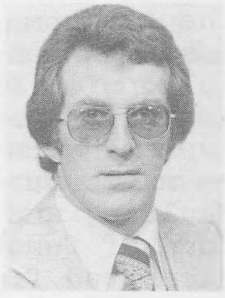
SEMINAR: "Statistics and Financial Modelling" - Toronto July 26,27.

NEW FACES AT SHARP

David Saunders



Morgan Smyth



Clive Edwards

David Saunders, Marketing Manager, U.K.
David joined Sharp after having worked for 7 years in management sciences (data processing & operations research). He has used APL extensively in production and financial planning.

J. Morgan Smyth, Toronto Branch Manager.
Well known in the APL community, Morgan is the author of two books on the development and promotion of APL. He has been involved in the design and development of several large computer systems.

Clive Edwards, Toronto Sales Manager.
Clive has joined the Toronto branch after 9 years in APL marketing and education. During that time he was responsible for the development and implementation of systems in inventory management, planning and forecasting, and an on-line money market accounting system. He will be assisting Morgan Smyth in developing the APL market in Toronto.

Jane Yates, Marketing and Support, Calgary.
Since 1973 Jane has been responsible for economics applications in APL such as project analysis, corporate modelling, and, most recently, a project which developed a land lease records maintenance system. Jane started at Sharp on June 1st.

APL PRESS

APL PRESS is a new publishing house devoted exclusively to APL. Its first book, to appear this summer, is a high school text on elementary analysis by K.E. Iverson. Several titles are planned for publication this year, and further manuscripts are being sought.

A newsletter is also planned, to present brief articles, problems, definitions of functions, reports on conferences, correspondence, and other items of interest to the APL community. The first issue, which is scheduled for July, will include a report by Professor Jenkins on a recent APL implementors workshop, an article on magic cubes by Professor Mauldon, and material on a new form of function definition excerpted from a forthcoming book.

Readers interested in receiving the newsletter and information on other publications, or in submitting material for publication, should write to APL PRESS,
Box 27,
Swarthmore,
Pennsylvania 19081.

More

STARK & CLEVER
APL

Reprinted from:

COMPUTER LIB

by permission.

Copies of the book are $7.00 postpaid from

Hugo's Book Service,
Box 2622,
Chicago, Illinois 60690.

Few people know all of APL, or would want to.
The operations are diverse and obscure,
and many of them are comprehensible only
to people in mathematical fields.
However, if you know a dozen or so you can
really get off the ground.

As in BASIC, you can use subscripts to get at specific elements in arrays. Referring to the examples above, if you type

JOE [2]

you get back on your typewriter its value

7.1

and if you type

NORA [2,4]

you get back

d

There are basically four kinds of information used by APL, and all of them can be put in arrays. Three of these types are numerical, and arrays of them look like this on paper:

Integer arrays: 2 4 -6 8 10 2048

Scalar arrays: 2.5 -3.1416 0.001 2795333.1
(a scalar is something that can be measured on a ruler-like scale, where there are always points in betweeen.)

Logical arrays: 1 0 0 0 1 0 1
(these arrays of ones and zeroes are called "logical" for a variety of reasons; in this case we could call them "logical" simply because they are used for picking and choosing and deciding.)

These three numerical types of information may be freely intermixed in your arrays. One more type, however, is allowed. It's hard to figure out from the manuals, but evidently this type can't be mixed in with the others too freely. We refer to the alphabetical or "literal" array, as in

The quick brown fox jumped over the lazy dog.

Now, pre-written APL programs can print out literal information, and accept it from a user at a terminal, which is why APL is good for the creation of systems for naive users (see "Good-Guy Systems," p. 13).

Literal vectors may be picked apart, rearranged and assembled by all the regular APL operators. That's how we twiddle our text.

CRASHING THE SYMBOLS TOGETHER

Now that we know about the operators and the arrays, what does APL do?

It works on arrays, singly and in pairs, according to those funny-looking symbols, as the APL processor scans right-to-left.

IVERSON'S TAFFY-PULL

A number of basic APL operators help you stretch, squish and pull apart your arrays. Consider the lowly comma (called "ravel," which means the same as "unravel").

,A	forget A's old dimensions, make it one-dimensional.
A,B	make A and B one long one-dimensional array.

Here is how we make things appear and disappear. ("Compresaion.")

A/B	A must be a one-dimensional array of ones and zeroes. The result is those elements of B selected by the ones. Example: 1 0 1 / c a t results in c t

The opposite slash has the opposite effect, inserting extra null elements where there are zeroes:

1 1 0 1\3 5 9
results in
3 5 0 9

Here's another selector. This operator takes the first or last few of A, depending on size and sign of B:

B ↑ A

and B ↓ A is the opposite.

If you want to know the relative positions of numbers of different sizes in a one-dimensional array,

⍋ (name of array)

will tell you. It gives you the positions, in order of size, of the numbers. And ⍒ does it for descending order.

These are just samples. The list goes on and on.

SAMPLE PROGRAMS

Here is an APL program that types out backwards what you type in. First look at the program, then the explanation below.

```
∇ REV
[1]  I ← ⍞
[2]  ⍞ ← ⌽ I
∇
```

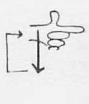
Explanation. The down-pointing triangles ("dels") symbolize the beginning and end of a program, which in this case we have called REV. On Line 1, the "Quote-Quad" symbol (on the right) causes the APL processor to wait for alphabetical input. Presumably the user will type something. The user's line of input is stuffed into thing or array 1. The user's carriage return tells the APL processor he has finished, so it continues in the program. On the second line, APL takes array I and does a one-sided ⌽ to it, which happens to mean turning it around. Left-arrow into the quote-quad symbol means print it out.

Because of APL's compactness, indeed, this magnificent program can all go on one line:

```
∇ REV
[1] ⍞ ← ⌽ I ← ⍞
∇
```

First the input goes into I, then the processor does a ⌽ I (reversal) and puts it out.

And here is our old friend, the fortune-cookie prisoner.



```
∇ INF
[1] ⍞ ← 'HELP, I AM CAUGHT IN A LOOP'
[2] → 1
∇
```

On line 1 the program prints out whatever's in quotes. And line 2 causes it to go back and do line 1 again. Forever.

THE TEST-AND-BRANCH IN APL

It should be mentioned at this point that branching tests are conducted in APL programs by specifying conditions which are either true or false, and APL's answer is 1 if true, 0 if false. (This is another thing these logical arrays are for.)

Example:

```
      3>2
```

This operation leaves the number 1, because 3 is greater than 2. So you could branch on a test with something like

```
→7×A>B
```

which branches to line 7 in the program if A is greater than B, and is ignored (as an unexecutable branch to line zero) if B is greater than A.

Some love it, some hate it.

THE APL ENVIRONMENT

Aside from the APL language itself, to program in APL you must learn a lot of "system" commands, alphabetical commands by which to tell the APL processor what you want to do in general -- what to store, what to bring forth from storage, and so on.

Ordinarily you have a workspace, a collection of programs and data which you may summon by name. When it comes-- that is, when the computer has fetched this material and announced on your terminal that it is ready-- you can run the programs and use the data in your workspace. You can also have passwords for your different workspaces, so others at other terminals cannot tamper with your stuff.

This is not the place to go into the system commands. If you're serious, you can learn them from the book or the APL salesman.

There are many, many different error messages that the APL processor can send you, depending on the circumstances. It is possible to make many, many mistakes in APL, and there are error messages for all of them. All of them, that is, that look to the computer like errors; if you do something permissible that's not what you intended, the computer will not tell you.

But it is a terminal language, designed to help people muddle through.

Good luck!

APL FOR USER-LEVEL SYSTEMS

(See "Good-Guy Systems," p. 15)

Because APL can solicit text input from a user and analyze it, the language is powerful for the creation of user-level environments and systems-- with the drawback, universal to all IBM terminals, that input lines must end with specific characters. In other words, it can't be as fully interactive as computer languages that use ASCII terminals.

Needless to say, the mathematical elegance and power of the system is completely unnecessary for most user-level systems. But it's nice to know it's there.

APL is probably best for systems with well-defined and segregated files-- "array-type problems," like payroll, accounts and so on. It is not suited for much larger amorphous and evolutionary stuff, the way list languages like TRAC are. Don't use APL if you're going to store large evolving texts or huge brokerage data bases, like what tankers are free in the Mediterranean.

The quickest payoff may lie in using APL to replace business forms and hasten the flow of information through a company. A salesman on the road with an APL terminal, for instance, can at once enter his orders in the computer from the customer's office, checking inventory directly. If the program is up.

IVERSON'S STRANGE AND WONDERFUL CHOICES OF SYMBOLS

Iverson's notation is built around the curious principle of having the same symbols mean two things depending on context. (Goodness knows he uses enough different symbols; doubling up at least means he doesn't need any more.) It turns out that this notation represents a consistent series of operations in astounding combinations.

The overall APL language, really, is the carrying through of this notation to create an immensely powerful programming language. The impetus obviously came from the desire to make various intricate mathematical operations easy to command. The result, however, is a programming language with great power for simpler tasks as well.

Now, the consequences of this overall idea were not determined by God. They were worked out by Iverson, very thoughtfully, so as to come out symmetrical-looking and easy to remember. What we see is the clever exploitation of apparent but inexact symmetries in the ideas. Often APL's one-sided and two-sided pairs of operators are more suggestively similar than really the same thing.

When Iverson assigns one-sided and two-sided meanings to a symbol, often the two meanings may look natural only because Iverson is such an artist. Example:

two-sided	one-sided
A×B	×B
A times B	the sign of B

This makes sense. To argue that it is inherent in "taking away half the idea of multiplication," however, is dubious.

Some symmetries Iverson has managed to come up with are truly remarkable. The arrow, for instance. The left arrow:

```
A←B
```

Assignment statement: B (which may have been computed during the leftward scan) is assigned the name of A;

and the right arrow:

```
→B
```

The jump statement, where B (which may have been computed during the leftward scan) is a statement number; the program now goes and executes that line.

This symmetry is mystically interesting because the assignment and jump statements are so basic to programming.

Or consider this:

```
□←X
```

print X.

```
X←□
```

take input from the user and stuff it into X.

Another weird example: supposedly the conditional branch

```
→B/A
```

(one way of writing, "jump to A if B is true") is a special case of the "compression" operator. (Berry 360 primer, 72 and 165.) This is very hard to understand, although it seems clear while you're reading it.

On the other hand, there is every indication that APL is so deep you keep finding new truths in it. (Like the above paragraph.) The whole thing is just unbelievable. Hooray for all that.

ROUND (an obscure and donnish joke)

ρ, the Greek letter "rho," is an APL operator for testing the size of arrays. When used in the one-aided format, it gives the aizes of each dimension of an array.

Thus

ρA, when A is $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

is 2 2.

And now

ρ'YOUR BOAT'
equals 9, since there are 9 letters in the array 'YOUR BOAT';

ρρ'YOUR BOAT'
is 1,
since ρ 9 is 1, and

ρρρ 'YOUR BOAT'
is likewise 1.

Thia language ia superb for "scientific" programming, including heavy number crunching and experimentation with different formulas on small data bases. (Big data bases are a problem.)
It is also not bad for a variety of simple business applications, such as payroll, accounting, billing and inventory.

FAST ANSWERBACK IN APL

If you want quick answers, the APL terminal just gives you the result of whatever you type in. For instance,

3 × 4

will cause it to print out

12

and the same goes for far less comprehensible stuff like

7 ≥ ⍋ ⍎ ? 1 2 3 4 (carriage return)

typed-in array

PROGRAMS IN APL

But the larger function of APL is to create programs that can be stored, named and carried out at a later time.

For this, APL allows you to define programs, a line at a time. The programs remain stored in the system as long as you want. Using the "Del" operator (∇), you tell the system that you want to put in a program. Del causes the terminal to help you along in various ways.

A nice feature ia that you can lock your APL programs, that is, make them inaccessible and unreadable by others, whether they are programmers or not. In this case you define a program starting with the mystical sign del-tilde (⍫) instead of del (∇), and invoke the names of dark spirits.

APL, like BASIC, can be classed as an "algebraic" language-- but this one is built to please real mathematicians, with high-level stuff only they know about, like Inner and Outer Products.
Paradoxically, this makes APL terrific for teaching these deeper mathematical concepts, helping you see the consequences of operations and the underlying structure of mathematical things. Matrix algebra, for instance, can be visualized a lot better by working up to it with lesser concepts (like vectors and inner products) enacted on an APL terminal.
It would be really swell if someone would put together a tour-guide book of higher mathematics at the grade/highschool level for people with access to APL.
Interestingly, Alfred Bork (U. of Cal. at Irvine) is taking a similar approach to teaching physics, using APL as a fundamental language in his physics courses.

WHERE TO GET IT

IBM doesn't sell APL services. Their time-sharing APL is available, however, from various suppliers. Of course, that means you probably have to have an IBM-type terminal, unless you find a service that offers APL to the other kind-- an addition which seems to be becoming fashionable.

Usual charge is about ten bucks an hour connect charge, plus processing, which depends on what you're doing. It can easily run over $15 an hour, though, and more for heavy crunching or printout, so watch it.

The salesman will come to your house or office, verify that your terminal will work (or tell you where you can rent one), patiently show you how to sign on, teach you the language for maybe an hour if he's a nice guy, and proffer the contract.

→APL services are probably safer to sign onto, in terms of risked expenses, than most other time-sharing systems. (Though of course all time-sharing involves financial risk.) Because the system is restricted only and exactly to APL, you're not paying for capabilities you won't be using, or for massive disk storage (which you're not allowed in most APL services anyway), or for acres of core memory you might be tempted to fill.

→ In other words, APL is s comparatively straight proposition, and highly recommended if you have a lot of math or statistics you'd like to do on a fairly small number of cases. Also good for a variety of other things, though, including fun.

Different vendors offer interesting variations on IBM's basic APL\360 package, as noted below. In other words, they compete with each other in part by adding features to the basic APL\360 program, vying for your business. Each of the vendors listed also offers various programs in APL you can use interactively at an IBM-type terminal, in many cases using an ordinary typeball and not seeing the funny characters; though how clear and easy these programs are will vary.

And remember, of course, that you can do your own thing, or have others do it for you, using APL.

APL is also available on the PDP-10, and presumably other non-IBM big machines.

Power and simplicity do not often go together.
APL is an extremely powerful language for mathematics, physics, statistics, simulation and so on.
However, it is not exactly simple. It's not easy to debug. Indeed, APL programs are hard to understand because of their density.
And the APL language does not fit very well on minis.

APL is not just a programming language. It is also used by some people as a definition or description language, that is, a form of notation for stating how things work (laws of nature, algebraic systems, computers or whatever).

For instance, when IBM's 360 computer came out, Iverson and his friends did a very high-class article describing formally in APL just what 360s do (the machine's architecture). But of course this was even less comprehensible than the 360 programming manual.

Falkoff, A.D., K.E. Iverson and E.H. Sussenguth. "A Formal Description of System/360," IBM Systems Journal, v.3 no. 3, 1964.
The formal description in APL.

IBM System/360 Operating System: Assembler Language. Document Number C28-6514-X (where X is a number signifying the latest edition). IBM Technical Publications, White Plains New York.
The Manual.

TERMINALS

For an APL terminal, you might just want a 2741 from IBM (about a hundred a month, but on a year contract).

Or see the list under "Terminals" (p.⍋), or ask your friendly APL company when you sign up.

Two more APL terminals, mentioned here instead of under "Terminals" for no special reason:

Tektronix offers one of its greenie graphics terminals (see flip side) for APL (the model 4013). This permits APL to draw pictures for you. It seems to be an ASCII-type unit.

Computer Devices, Inc. supposedly makes an an APL terminal using the nice NCR thermal printer, which is much faster and quieter than a mechanical typewriter. Spookier, though. And the special paper costs a lot of money.

BIBLIOGRAPHY

Iverson has a formal book. Ignore it unless you're a mathematician: Kenneth E. Iverson, A Programming Language. Wiley, 1962.
Paul Berry, APL\360 Primer, Student Text. Available "through IBM branch offices," or IBM Technical Publications Department, 112 East Post Road, White Plains, NY 10601. No IBM publication number on it, which is sort of odd. 1969.
→This is one of the most beautifully written, simple, clear computer manuals that is to be found. Such a statement may astound readers who have seen other IBM manuals, but it's true.
A.D. Falkoff and K.E. Iverson, APL\360 Users' Manual. Also available from IBM, no publication number.
POCKET CARDS (giving very compressed summaries) are available from your APL supplier

Paul Berry. APL\1130 Primer. Adapted from 360 manual. Same pub. But for version of APL that runs on the IBM 1130 minicomputer.

What you really need to get started is Berry's Primer, Falkoff and Iverson's manual, and a pocket card. Plus of course the system and the friend to tutor you.

Sharp Special Systems

Along with Sharp's involvement with and expertise in APL, the company also has considerable experience in the design and implementation of minicomputer-based systems for process control, data processing and telecommunications applications. A good example is the packet switching network discussed earlier.

Known as 'Special Systems', these activities are discussed in the newsletter in the hope that they will be of general interest.

THE EVOLUTION OF THE MINI MARKET

by Hugh O'Rourke

In the very inflation-conscious atmosphere of the last few years, the cost of computers, especially mini-computers, is often cited as an example of falling costs. And it is quite true that economies of volume and technology have brought prices down. A small processor with 16K words of core and a teletype that sold for around $35K seven or eight years ago can now be had for well under $10K. Unfortunately, however, the publicity which this rather dramatic price drop has received, has created a somewhat unrealistic impression among prospective small-computer buyers, and has added a new dimension of headaches to marketing small computer systems.

Until the last few years, small computer systems were exactly that - small computer systems. They generally consisted of a processor with from 4 to 16K of core, a teletype with paper reader/punch attached, and one or more interfaces to some sort of processing, monitoring or communications gear. The programming was invariably done in a machine-level language (Assembler) - and typical implementations took six months and cost between $50K and $70K for hardware and software. The line between small and large computer systems was clearly drawn.

Today, the situation is quite different. A typical mini-computer system is neither small nor inexpensive. Although the cost and size of the processors has decreased, the amount of core memory and peripheral devices has increased. In an effort to widen the applicability of mini-computers, the manufacturers have developed interfaces for an endless array of peripherals such as medium and large capacity disks, fast line-printers, industry-compatible magnetic tape drives, punched card readers and so on, with the result that mini-computer system price tags are seldom less than $100K for hardware alone.

The scene is further complicated by the spread of those "other" so-called higher-level languages. Cobol on a PDP-8? Well, truth is stranger than fiction. And although I'm sure it helps the manufacturers to sell minis, I suspect that some of the impetus for this has come from owners of small computer systems who, after the dust has settled and the troops departed, have been left to cope with cores-

full of Assembler programs written with more individuality than organization. In any event, the problem with higher level languages is that they escalate hardware costs - more core, special gadgets to cope with core that goes further than 16 bits, compiler and operating system licence fees, and so on.

With all this, buyers still expect mini-computer systems to be inexpensive, and it is very difficult to convince them that, although the computer would fit into a briefcase, the computer system to run the payroll is going to cost $175K for hardware and software.

There are, nonetheless, more and more minis being sold every year. The growth rate of companies like D.E.C. and Datagen has been phenomenal, with a corresponding increase in the growth of systems houses. In particular, "specialty" system houses have been emerging - firms that specialize in pre-hatched special-purpose systems for, say, payroll or inventory control. Step right up, folks, get your handy-dandy payroll system, one all-inclusive price, just plug it in and watch it print those cheques, folks.

Seriously, though, the problem firms like Sharp are finding now is that, without changing ourselves, we have gone from being the few firms that specialize in small computer systems to firms that specialize in custom-programmed small computer systems. Remember the good old days when all minis were custom programmed?

So, how to stay in the market place? How to reduce the selling price of the hardware, or the software, or both? For the hardware, significant economies can be realized by purchasing cheaper peripherals. Not many peripherals are actually manufactured by mini-makers so they can usually be bought for less elsewhere. But the drawback is service. Mini mainframe makers are understandably reluctant to service peripherals you don't buy from them and few users are willing or able to cope with more than one service contract. The answers are either to offer service ourselves, or find a solid, reliable local service firm that specializes in "grab-bag" systems - but both of these are easier said than done.

That leaves software, and what we have begun to do in the last while is a little "pre-packaging" ourselves. One of the more notable items is a real-time systems executive (REALSEX) for the PDP-11 which has been developed, refined and standardized over the past few years by Bob Johnston. But developing items like this is a slow process when it is only being done concurrently with a system implementation for a customer - and there is virtually no chance of doing any applications packages. It seems to me that we have to work towards being able to sell mini systems of which 60-80% of the software is bolted together from a library of pre-programmed modules and programs.

The spectrometer readout systems developed in Carleton Place and the mini-based Inspector system development in Rochester are two good examples of the kinds of things we should be doing. But I think it's fairly obvious that we should be doing more.

Suite 1400, 145 King Street West, Toronto, Canada M5H 1J8

Update

☐ Please amend my mailing address as indicated.
☐ Add to your mailing list the following name(s).
☐ Send me SHARP APL manuals and product literature as listed.

☐ Note my comments: _____

The Newsletter is a regular publication of I.P. Sharp Associates Limited. Contributions and comments are welcomed and should be addressed to: Jeanne Gershater, Editor, I.P. Sharp Newsletter, Suite 1400, York Centre, 145 King Street West, Toronto, Ontario, M5H 1J8.

I.P. Sharp Associates Limited Head Office: Suite 1400, 145 King St. West, Toronto, Canada M5H 1J8 (416) 364-5361

Canada — Regional Offices

Calgary
Suite 1000,
615 — 2nd Street S.E.,
Calgary, Alberta
T2G 4T8
(403) 265-7730

Edmonton
Suite 505,
10065 Jasper Ave.,
Edmonton, Alberta
T5J 3B1
(403) 424-7395

London
Suite 510,
220 Dundas Street,
London, Ontario N6A 4R2
(519) 434-2426

Montreal
Suite 1610,
555 Dorchester Blvd. West,
Montreal, Quebec
H2Z 1B1
(514) 866-4981

Ottawa
Suite 600,
265 Carling Ave.,
Ottawa, Ont.
K1S 2E1
(613) 236-9942

Vancouver
Suite 604,
1112 West Pender St.,
Vancouver, B.C.
V6E 2S1
(604) 682-7158

England

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

London
118 - 119 Piccadilly,
Mayfair, London W1V 9FJ
England
(01) 629-1564

Reading
Intersystems (U.K.) Limited,
1A Buckland Road,
Reading, Berks
RG2 7SP
(0734) 863334

Europe

Intersystems, B.V.
Herengracht 244,
Amsterdam 1002,
The Netherlands
(020) 250401

APL Europe S.A.
c/o Arthur Young & Co.,
191 - 7 Boulevarde de Souverain,
Brussels, Belgium
649 94 30

I.P. Sharp GMBH
Kaiser-Friedrich-Ring 98
4000 Duesseldorf
West Germany
(0211) 579084

U.S.A. — I.P. Sharp Associates, Inc.

Boston
Suite 812,
148 State St.,
Boston, Mass. 02109
(617) 523-2506

Chicago
Suite 424,
8501 West Higgins Rd.,
Chicago, Ill. 60631
(312) 693-5895

Dallas
Suite 1148,
Campbell Centre,
8350 Northcentral Expressway,
Dallas, Texas 75206
(214) 369-1131

Minneapolis
Suite 202,
1 Appletree Square,
Bloomington, Minn. 55420
(612) 3749406

New York City
Suite 250, East Mezz.,
Pan Am Bldg.,
New York, N.Y. 10017

Newport Beach
Suite 1135,
610 Newport Centre Drive,
Newport Beach, Ca. 92660
(714) 644-5112

Rochester
Suite 1150,
183 Main Street East,
Rochester, N.Y. 14604
(716) 546-7270

San Francisco
Suite C409,
900 North Point Street,
San Francisco, Ca. 94109
(415) 673-4930

Seattle
Suite 3735,
SeaFirst Building,
1001 Fourth Ave.,
Seattle, Wa. 98154
(206) 938-0500

Washington D.C.
1815 Fort Myer Drive,
Arlington, Va. 22209
(703) 527-3333

White Plains, N.Y.
Suite 39, Station Plaza,
250 East Hartsdale Ave.,
Hartsdale, N.Y. 10530
(914) 472-6380

SHARP APL Local Access In:

Canada	U.S.A.	Europe
Calgary	Atlanta	London
Edmonton	Boston	Gloucester
Halifax	Buffalo	Amsterdam
Kitchener	Chicago	Duesseldorf
London	Cleveland	Paris
Montreal	Dallas	
Ottawa	Des Moines	
Quebec City	Ft. Lauderdale	
Saskatoon	Los Angeles	
Sault Ste. Marie	Miami	
Toronto	New York City	
Vancouver	Rochester	
Winnipeg	San Francisco	
	SantaAna	
	Seattle	
	St. Paul, Minn.	
	Syracuse	
	Washington	
	White Plains	

APL Operator (416) 363-2051