APL: Good For The Brain

A Programming Language has been around for many years, yet it is fast becoming THE new language. This exciting development in small computing systems is something that we think will be good for all of us.

IT HAS BEEN observed that not only does a language (English, French etc!) provide a means of expressing thoughts, it also tends to limit thoughts, to those expressible in the language whether to others, or to one's self. Just as a road provides a path for transportation, if you get used to travelling by car you kind of have to stick to the road, and the surrounding terrain remains undiscovered.

So it is with computer languages. Iverson Notation was developed for humans to use, to advance their ability to think and solve problems. APL grew from this, and is probably the only language to pick up an already developed notation, as opposed to acting as a compromise between English and machine code.

Thus while other languages tend frequently to constrict one in trying to solve a problem, learning and using APL tends to help one to handle the problem. In fact with experience one seems to be able to write an APL program while thinking up the method of solution. Other languages usually require an intermediate block diagram or flow chart stage for a similar problem. The proof of this lies in the absolutely fanatical following for APL, and the well attended and highly regarded APL conferences which take place to discuss applications and possible new features for the language.

We think APL is great for the mind and body, and that it's about to arrive in a big way. This article deals with what it is, where it's been, and a look at when it should be used, and where it's going.

WHAT IT'S LIKE

In as much as APL is quite different from other computer languages, has vastly different qualities, and has fanatical supporters, it is very useful to know something about the language itself.

APL having originally evolved from Iverson Notation, it is most helpful to start with those features which are actually part of the notation.

IVERSON NOTATION

The first step to getting a grasp on Iverson Notation is to be familiar with a few 'buzz-concepts' essential to the topic.

Essentially Iverson Notation is a mathematical notation which permits expressions to be written in very compact and condensed forms. The idea is that by giving a lot of the most used functions single symbol names (instead of just + – x and ÷) the user can not only more easily write his ideas down, but even learn to think on a higher level. A good example of this is a sorting operation, for example putting a list of names in alphabetical order. Iverson Notation gives you the concepts of how to think of the list of names, and then allows you to write down, using just a couple of symbols, the method used to do it. Example later.

DATA TYPE

In a mathematical expression, one may have a variety of different types

APL Applied

APL's big strength stems from the ability of a person familiar with the language to write a program as fast as thinking of the method. In fact one even tends to think of the solution in APL form. The other influential feature is that APL is an interactive language.

These two combine to produce an ideal way to use a computer for applications where the programs need to be ready fast, and where the applications area may be new (and hence the program is experimental and may require much revision). This is not to say that in other applications APL is not suitable, but until recently APL has been thought of as expensive in terms of execution time and memory use, or even simply to obtain access. In addition compared to other languages there are relatively few experienced APL programmers, the number familiar with the language will of course grow.

One estimate holds that an APL programmer can produce a program on the order of 25 times the speed of a COBOL programmer, (including debugging) and that the result is of course a far more compact listing.

One point to watch, however, is that while almost anyone can plough though a BASIC, FORTRAN or COBOL program (with enough patience), it is possible to program in APL in such a dense and tricky manner as to make the program virtually impossible to decipher by other persons. Programmers who do this also enjoy reducing 100 line FORTRAN programs to an APL one-liner. (There seems to be some special attraction to the 'one-line' challenge!)

However, if a program is written with reading also in mind, with a bit of descriptive documentation, there is no problem.

What APL Looks Like

SCALAR DYADIC FUNCTIONS

X+Y		X plus Y
X-Y		X minus Y
X×Y		X times Y
X÷Y		X divided by Y
X⋆Y		X to the Y-th power
X⌈Y		Maximum of X and Y
X⌊Y		Minimum of X and Y
X∣Y		X-residue of Y
X⍟Y		Base-X logarithm of Y
X!Y		Binomial coefficient: Y items taken X at a time
X○Y		See trigonometric functions
X<Y	•	X less than Y
X≤Y	•	X less than or equal to Y
X=Y	•	X equal to Y
X≥Y	•	X greater than or equal to Y
X>Y	•	X greater than Y
X≠Y	•	X not equal to Y
X∧Y		X and Y
X∨Y		X or Y
X⍲Y		Not both X and Y
X⍱Y		Neither X nor Y

TRIGONOMETRIC FUNCTIONS

R←X○Y

(Y in radians)		(R in radians)	
X	R	X	R
0	(1-Y⋆2)⋆.5		
1	Sine Y	¯1	Arcsin Y
2	Cosine Y	¯2	Arccos Y
3	Tangent Y	¯3	Arctan Y
4	(1+Y⋆2)⋆.5	¯4	(¯1+Y⋆2)⋆.5
5	Sinh Y	¯5	Arcsinh Y
6	Cosh Y	¯6	Arccosh Y
7	Tanh Y	¯7	Arctanh Y

SCALAR MONADIC FUNCTIONS

+Y		Y
-Y		0-Y
×Y		Signum Y
÷Y		Reciprocal of Y
⋆Y		e to the Y-th power
⌈Y	•	Ceiling of Y
⌊Y	•	Floor of Y
∣Y		Absolute value of Y
⍟Y		Natural logarithm of Y
!Y		Factorial Y; Gamma Y+1
○Y		Pi times Y
?Y	#	A random number from ⍳Y
~Y		Not Y

• ⎕CT dependent
⎕IO dependent

SYMBOLS

( )	Parentheses for nesting
[ ]	Brackets fnr indexing
⎕	Quad for input-output
⍞	Quote-quad for character input
'	Quote delimits character literals
⍝	Lamp indicates comment
¯	Negative sign
E	Exponential notation
∆	Delta. Trace (T∆) and stop (S∆) control
⌶	I-beam for system functions
.	Decimal point
^	Caret locates errors
;	Semicolon separates list elements
⋄	Diamond used as statement separator
→	Branch
:	Colon delimits labels and locks
⌷	Squish-quad display of non-printable characters

The following characters are available as graphics:

$ α { ⊢ ⊂ ∩ ¨
¢ ω } ⊣ ⊃ ∪ _

FUNCTION DEFINITION

Function definition is opened or closed bv ∇.
Use of ⍫ at the close will lock the function.
The following are valid with open definition.

[⎕]	Display entire function
[n⎕]	Display line n
[⎕n]	Display from line n to the end
[n⎕p]	Display line n and position at p prior to editing
[n⎕0]	Display line n and position at right-hand end of line
[n]	(Re)define line n

MIXED FUNCTIONS

XρY		Reshape Y to have dimensions of X
ρY		Dimensions of Y
X[Y]	#	Y-th elements of X
X⍳Y	•#	First locations of Y within vector X
⍳Y	#	Y consecutive integers from origin (0 or 1)
X∊Y	•	Membership of X in Y
X⊤Y		Representation of Y in number system X
X⊥Y		Value of the representation Y in number system X
X?Y	#	X integers selected randomly without repetition from ⍳Y
X⌽[Z]Y		Rotation by X along the Z-th dimension of Y
⌽[Z]Y		Reversal along the Z-th dimension of Y
X⍉Y	#	Transpose of Y according to X
⍉Y		Transpose of Y (⌽⍳ρρY)⍉Y
X,[Z]Y		Catenate or laminate Y to X along Z-th dimension
,Y		Ravel of Y (makes Y a vector)
X↑Y		Take first or last X elements of Y as X is + or -
X↓Y		Drop first or last X elements of Y as X is + or -
X←Y		X is assigned the value of Y
⍋Y	#	Index vector such that Y[⍋Y] is in ascending order
⍒Y	#	Index vector such that Y[⍒Y] is in descending order
X/[Z]Y		X (logical) compressing along the Z-th dimension of Y
X\[Z]Y		X (logical) expanding along the Z-th dimension of Y
R←X⌹Y		Matrix divide. X approximates R+.×Y
⌹Y		Generalized matrix inverse of Y
⍎Y		Executes the character string Y
⍕Y		Character representation of Y
X⍕Y		Numeric expression Y is converted to character according to numeric format control vector X

● below denotes any scalar dyadic function

●\[Z]Y	● applied cumulatively along the Z-th dimension of Y
●/[Z]Y	The ● reduction along the Z-th dimension of Y
X∘.●Y	Generalized outer product of X and Y
X●.●Y	Generalized inner product of X and Y e.g. X+.×Y is ordinary matrix product of X and Y
⊖	⌽ applied to first dimension (reverse or rotate)
⌿	/ applied to first dimension (compress or reduce)
⍀	\ applied to first dimension (expand or scan)

If the expression [Z] is omitted, the operation applies to the last dimension of the argument array Y. The expression [Z] is ⎕IO dependent.

• ⎕CT dependent
⎕IO dependent

SYSTEM COMMANDS

Note: Bracketed words are optional.

) [n]	Allow editing of immediate execution line at position n
)nnnnnnn [:lock]	Sign-on - invalid sign-on acts as)BLOT
)BLOT	Print a mask for typing secure information
)CLEAR	Clear the active workspace
)CONTINUE [HOLD] [:lock]	Terminate a session and store the active workspace in CONTINUE
)COPY name [:lock] [list]	Copy objects from a workspace
)DROP name [:lock]	Delete a stored workspace
)ERASE list	Erase objects
)FNS [letters]	List names of functions
)GROUP name list	Define a group
)GROUP name	Disperse a group
)GRP name	List members of a group
)GRPS [letters]	List names of groups
)KEYB LOCK	Lock keyboard (to receive messages)
)KEYB NOMSG	Suppress incoming messages
)KEYB	Return keyboard to normal state
)LIB [nnn]	List names of workspaces in library
)LOAD name [:lock]	Activate a copy of a stored workspace
)MSG taskid text	Send message to designated terminal
)MSGN taskid text	Send message to designated terminal without locking sender's keyboard to receive reply
)OFF [HOLD] [:lock]	Terminate a work session
)OPR text	Send message to SHARP APL operator
)OPRN text	Send message to SHARP APL operator without locking sender's keyboard to receive reply
)PCOPY name [:lock] [list]	As)COPY, but protect contents of active WS from being overwritten.
)PORTS aaa	List port number and task-ID of user aaa
)PORTS nnnnnnn	List port number and task-ID of user nnnnnnn
)RESET	Clear the)SI stack
)SAVE [name] [:lock]	Store a copy of the active workspace
)SEAL	Lock all functions in the workspace, prevent dispersal of contents
)SI	Display the state indicator
)SIV	Display state indicator and names of local variables
)SYMBOLS [n]	Display/set symbol table size
)TERM [name]	Display/set terminal type. See WS 5 TERM for current list of names
)VARS [letters]	List names of variables
)WSID [name]	Display/set workspace name

Fig. 1. I. P. Sharp Associates provide a handy reference card for programmers using their systems. It is just about all you would ever need to remember or take with you to a desert island equipped with only an APL computer, on a scant 224 square inches of card. No more heavy manuals to lug around! Most of the information in this exerpt is general, we have included the section on system commands to give an idea of what is typical. The special characters shown here are those produced on an impact kind of typewriter terminal.

of data. Most people are familiar with integers and real numbers, but also letters, words, collections of characters can be data.

Do not confuse, however, character data with names for variables. For example, in the ordinary algebraic expression A=2xB, A and B are variables while 2 is a constant. On the other hand, in a computer context it would be quite permissible for C='ABCD', ie 'ABCD' is the 'value' of variable C.

RANK

In ordinary algebra we mostly use variables of a single 'element', known as scalars (eg a=2). Sometimes vectors are used, really a list of numbers known by one variable name (eg a=3, 7, 2, -1, or a(1)=3, a(2)=7, a(3)=2 and a(4)=-1). 'Matrices' are expecially popular for such applications as solution of complicated equations, and have two 'subscripts'.

Example:

```
a = 1  3  0
    7  5  4
    8 -7  2
```

or a(1,1) = 1, a(2,1) = 7, a(3,3) = 2, etc.

The concept of using subscripts can be extended on and on to any number of 'dimensions'. The general term for all such variables is 'array'. The number of dimensions or subscripts, it has is known as its 'rank'. Hence a (b,c,d) has rank of 3, a 2 dimensional matrix has a rank of 2, a vector rank 1, and a scalar has rank 0.

FUNCTIONS

APL has a large number of characters which denote functions (examples are the familiar + — x ÷ but there are many more exciting ones! A couple of useful words to know here are 'monadic' and 'dyadic' which mean 'having one argument' and 'having two arguments' respectively. Note that each argument may be an array. Thus A+B may yield a scalar if A and B are scalar, but if they are arrays of the same size and shape, each element of A is added to the corresponding element of B to make the corresponding element of the resulting array.

CONSISTENCY AND EXECUTION ORDER

One of the great virtues of Iverson Notation is its consistency. All monadic

```
        ∇ACCUM[⎕]∇
      ∇ Z←ACCUM A;B;L
[1]    ⍝ IMPROVED
[2]     A←A[⍋A[;1];]
[3]     B←A[;1]
[4]     L←(1↓B)≠¯1↓B
[5]     L←L,1
[6]     B←L/B
[7]     A←L/+\A[;2]
[8]     A←A-0,¯1↓A
[9]     Z←B,[1.5] A
      ∇


        ∇CHARS[⎕]∇
      ∇ Z←CHARS X;A
[1]    ⍝ RETURNS A LIST OF CHARACTERS IN (X),
[2]    ⍝ SORTED ACCORDING TO THEIR POSITION IN ΔAV
[3]     A←' ',ΔCHAR
[4]     Z←A∊X
[5]     Z←Z/A
[6]     X←((X⍳X)=⍳⍴X)/X←(∼X∊A)/X
[7]     X←ΔAV⍳X,Z
[8]     Z←ΔAV[X[⍋X]]
      ∇
        ∇DEB[⎕]∇
      ∇ Z←DEB X
[1]    ⍝ DELETE LEADING, TRAILING AND EXTRA 1↑0⍴X IN (X)
[2]     Z←1↑0⍴X
[3]     →(1=⍴⍴X)⍴L1
[4]     X←,X
[5]    L1:Z←(-∼Z←¯1↑Z)↓X←(Z∨0,¯1↓Z←X≠Z)/X
      ∇
        ∇DLBDTB[⎕]∇
      ∇ Z←DLBDTB X
[1]    ⍝ REMOVES LEADING AND TRAILING BLANKS FROM TEXT INPUT (X)
[2]     Z←((∨\Z)∧⌽∨\⌽Z←X≠' ')/X←,X
      ∇


        ∇COMPRESSNAME[⎕]∇
      ∇ Z←COMPRESSNAME NAME;L
[1]    ⍝ AN ALGORITHM FOR NAME COMPRESSION
[2]    ⍝    [1] DELETE SECOND ELEMENT OF REPEATED CONSONANT PAIR
[3]    ⍝    [2] DELETE 'AEIOUY' EXCEPT WHEN FIRST LETTER IN NAME
[4]    ⍝ 1ST STATEMENT IS NOT ABSOLUTELY REQUIRED
[5]    ⍝
[6]     →(⍴NAME←DEB NAME)↓⍴Z←''
[7]     Z←∼L←NAME∊'AEIOUY'
[8]     L←(0,1↓L)∨NAME≠1↓NAME,' '
[9]     Z←L/NAME
      ∇
```

Fig. 2. Another example of what APL looks like, this printout from a DEC dot matrix terminal. The resolution is not quite as good, nor the resulting characters so exotic looking, but the terminal is faster and quieter.

functions are written with the one function symbol to the left of the argument, such as !F which means the factorial of F, ÷X which is the reciprocal of X, or ?Y, a random selection from the first. Y integers, while |A means absolute value of A. And remember F,X,Y and A could be arrays! In fact, the APL user tends to think that the basic data form is the array, which occassionally 'happens' to be a scalar or vector.

Similarly, all dyadic functions are written with the symbol between the two arguments.

Finally, execution order is right to left, regardless of functions (ie x does not take precedence over +). The order may be changed using brackets. This order is consistent with the normal use of functions such as sin, log, d/dx etc.

THE APL CHARACTER SET

In order to write all functions compactly, a large collection of new symbols was designed, which is now fossilized in the type elements of various impact type terminals. These usually have all the upper case letters, plus numbers and punctuation and so forth, and of course quite a selection of not normally found symbols. Even more symbols are made by backspacing and over striking a second symbol on the first. (eg A, ☐ and ÷ to make ⌹ etc).

Before non-APL fans get turned off by all this new symbology, it must be said that learning these symbols is extraordinarily easy, as many of them remind you of their meaning by their shape.

Two important notes to make: the use of the proper multiply sign (rather than an asterisk for multiply as in many languages) and ← instead of =. This is a logical choice since A←A+1 really means 'A is assigned the value A+1', rather than 0=1. (By the way, APLers often say 'gets' rather than 'is assigned the value').

In Fig. 1 is a listing of APL symbols and their meanings, while Fig. 2 shows what a typical output looks like and there are more annotated examples in Fig. 3.

ENVIRONMENT

The implementation of Iverson Notation as a computer language brought real muscle to the elegant notation. As a computer language it is used interactively, that is to say you sit at a terminal (or at your personal APL machine!) and communicate directly and immediately with the computer.

You have the option of typing a statement and obtaining its result on the spot (immediate execution mode), writing a function or program (function definition mode), and executing such a function or program (again — immediate execution mode). Various editing and line numbering facilities are provided for function writing, and diagnostics and error messages help to debug functions.

THE WORKSPACE

On 'big systems', each APL user has a 'workspace', a fenced-off piece of memory, as it were, of some arbitrary size, say 50k bytes, where all his work is stored, all variables functions and programs, and in which the programs are executed. On completion of a session the user can store his workspace, even including partially executed programs, as complete state information is contained in the workspace. This storing process is usually onto disk, and is done in a relatively fool proof way as the user signs off. The workspace then sits in suspended animation, awaiting the user's continued efforts next time.

A user may also copy functions from another workspace into his own, (if so permitted) which of course gives access to alot of useful software which others on the same system have developed.

OTHER BIG SYSTEM APL FEATURES

Just as a user may copy information from another workspace, so can he access system 'libraries' where large volumes of software are typically found, such as routines for plotting, various engineering, statistical, and work processing packages, and of course games.

Additionally, extensive file systems (usually disk) are generally available for storing large amounts of program code or data.

APL CHARACTERS ON EXISTING MACHINES

How do you put APL on a machine which has no APL character set? While new machines will be able to display the characters, and new character generator ROMs may satisfy a few other ways around the problem are possible.

One is to use three to five letter keywords instead of symbols, such that a single key could signify the function, and the whole keyword pops onto the screen. This doesn't provide an elegant looking display, but it does give you APL with only a small sacrifice.

IMPLEMENTATION ON MICROS

The workspace, library and file concepts are more system dependent than the Iverson Notation part of APL. They are also likely to be seen only in limited form at first on small systems. For instance, one workspace is likely to be almost the entire memory of a micro-based machine, so there will be only one user at a time. A disk unit will be essential, for storing workspace(s) and files, and one will presumably be able to create one's own libraries, or buy library disks with various useful packages on them.

'Virtual Workspace' schemes are already being worked on, wherein the user thinks he has a large amount of memory available, when in fact the machine has only a small amount, but swaps portions of the workspace on and off a disk as they are needed. This requires some fancy memory management to accomplish effectively.

```
      ∇ ENTRY
[1]     B←10↑⍞
[2]     A←A,B
[3]     →1↑B≠'*'
      ∇
```

ENTRY allows the user to enter a list of names, one at a time, alloting ten characters to each name, and filling any unused positions with blanks. ENTRY stops if the user types a single "*". The resulting vector is A.

```
      ∇ RESHAPE
[1]     C←((⍴A)÷10),10
[2]     D←C⍴A
      ∇
```

RESHAPE changes the list of names from a vector to an array with one name per row.

```
      ∇ ORDER
[1]     E←ALPHA⍳D
[2]     L←1↑C
[3]     F←E,⍳L
[4]     N←10
[5]     F←F[⍋F[;N];]
[6]     N←N-1
[7]     →5+3×N=0
[8]     D←D[F[;11];]
      ∇
```

ORDER takes the array D, makes a numerical array E whose entries represent the letters in D, according to ALPHA. E is sorted by row, and the new order is then imposed on D. Job finished!

Below, the functions in action.

```
        ALPHA
 ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

ALPHA is the vector variable to which we assigned the desired organizing sequence. It's value is the character "blank" followed by the alphabet. Other orders or characters could have been chosen instead.

```
        ENTRY
JOHN
FRED
ANDY
*
```

User types "ENTRY", the the function waits for the users entries, until it encounters the "*".

```
        A
JOHN      FRED      ANDY      *
        B
*
```

The resulting A and B values after ENTRY.

```
        RESHAPE
        C
4  10
        D
JOHN
FRED
ANDY
*
```

RESHAPE, and the resulting C and D values.

```
        ORDER
        E
 11  16   9  15   1   1   1   1   1   1
  7  19   6   5   1   1   1   1   1   1
  2  15   5  26   1   1   1   1   1   1
 28   1   1   1   1   1   1   1   1   1
        L
4
        F
  2  15   5  26   1   1   1   1   1   1   3
  7  19   6   5   1   1   1   1   1   1   2
 11  16   9  15   1   1   1   1   1   1   1
 28   1   1   1   1   1   1   1   1   1   4
        D
ANDY
FRED
JOHN
*
```

User executes the function ORDER, and the resulting values of E, L, F, and finally what we were waiting for . . . D, now in order.

```
      ∇ EXAMPLE
[1]     A←''
[2]     ENTRY
[3]     RESHAPE
[4]     ORDER
[5]     ⎕←D
      ∇
```

EXAMPLE is a function which ties these three functions together into one, and then prints the result out.

Fig. 3. An example of how to sort a list of names. The 3 functions are listed at the top. Below is a series showing what the user did (indented), and what the machine's response was. (Yes, we know it could be done in one line or less).

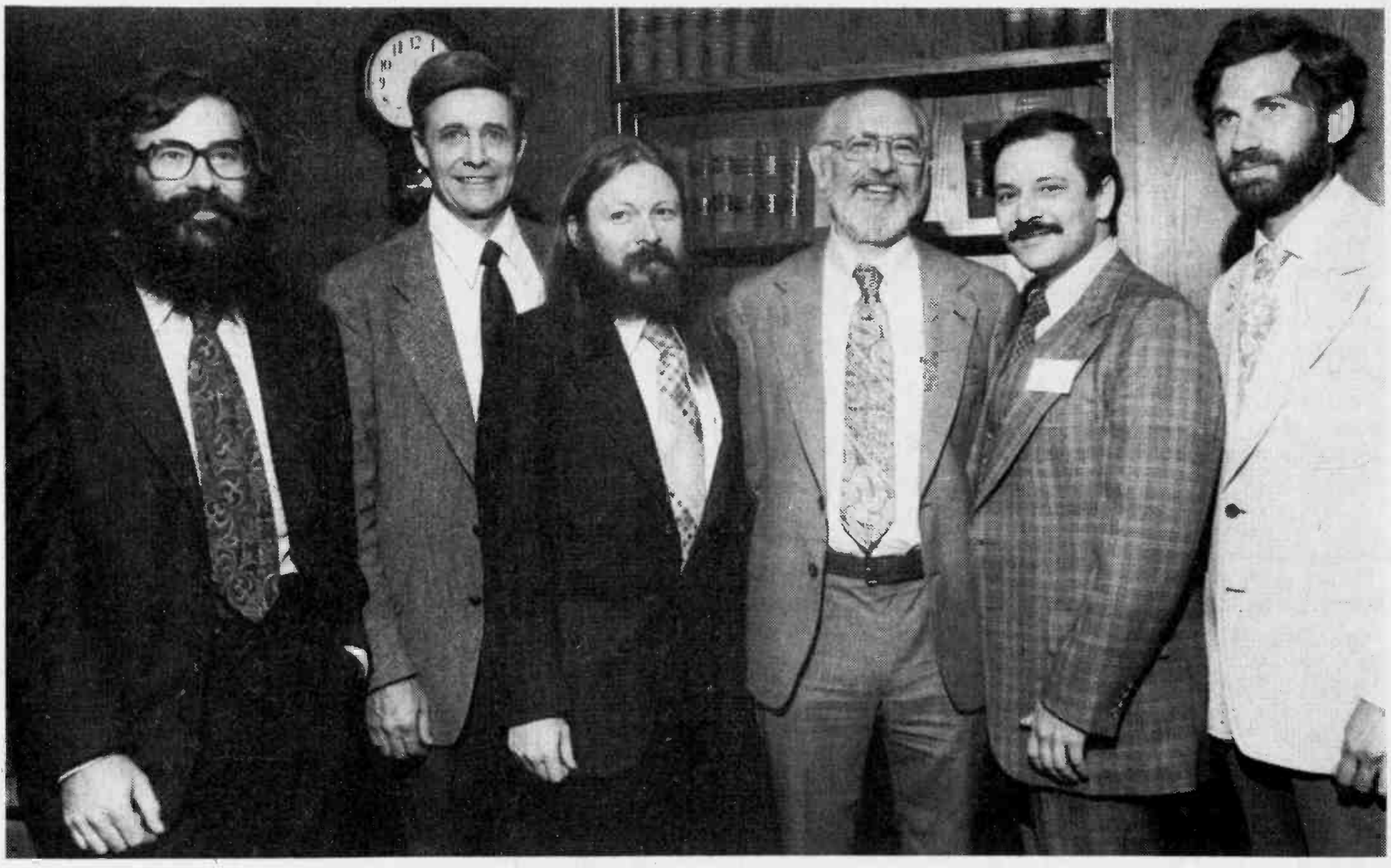


Fig. 4. The caption originally read: "This historic photograph was taken in the hospitality suite at the recent APL users meeting. From left to right are Dick Lathwell, Ken Iverson, Roger Moore, Adin Falkoff, Phil Abrams and Larry Breed. It is believed to be the first time that all six 'originators of APL' have been in the same place at the same time, it is probably the first time that all six have worn jackets and ties simultaneously and the first time that Ken and Adin have been observed to smile simultaneously." Reproduced by kind permission from The I. P. Sharp Newsletter.

Where has APL Been?

Not hiding, just expensive?

It was in the late 1950's that Kenneth Iverson, a Canadian, at that time a professor at Harvard University, developed his notation system. Iverson Notation was intended for analyzing and communicating problems in information processing which he and his students were involved with.

Iverson and IBM joined forces in 1960, and there, with the help of those fellows in Fig. 4, IBM's APL interpreter resulted.

For a long time, APL was only available to those with access to large computers, which made it expensive to get into, but the enthusiasm of those who could use it was extrordinary. So much so, in fact, that I. P. Sharp Associates now provide addicts in Europe, North America, Scandinavia and Australia with access to APL on their Toronto systems via a vast communications network.

What was needed was a stand-alone type machine which was inexpensive, in order for APL to be worthwhile for the smaller user, or even as an educational tool. Steps in that way were taken with IBM's 5100 series (a dedicated APL machine which pretended to be a 360 cpu and thus could run the 360 APL interpreter unchanged, including bugs!), also machines from Hewlett Packard and DEC.

THE MCM MACHINE

In what might seem like attempting to win the Indianapolis 500 on roller skates, MCM Computers, (a Canadian company!) in 1974 introduced the MCM70 a desktop APL computer with one line alphanumeric display, keyboard, dual cassettes and was based on the 8008! It might be obvious these days that microprocessors are the way to go but at the time development started on that MCM machine people were skeptical of seeing any kind of high level language on a micro, let alone APL, and especially on the 8008! More on this in 'The MCM Story'.

RECENT MICRO MOVES

Until recently there hasn't been much stirring in the way of APL for the microcomputer system, in the way BASIC has been available for every microcomputer around.

Now, suddenly there appears to be a terrific battle brewing, as the race is on to feed APL to what is expected to be an open-mouthed market. Judging by the almost astounding enthusiasm of current APL users, each new user will be spreading the word far more quickly than was true with BASIC, that is if he

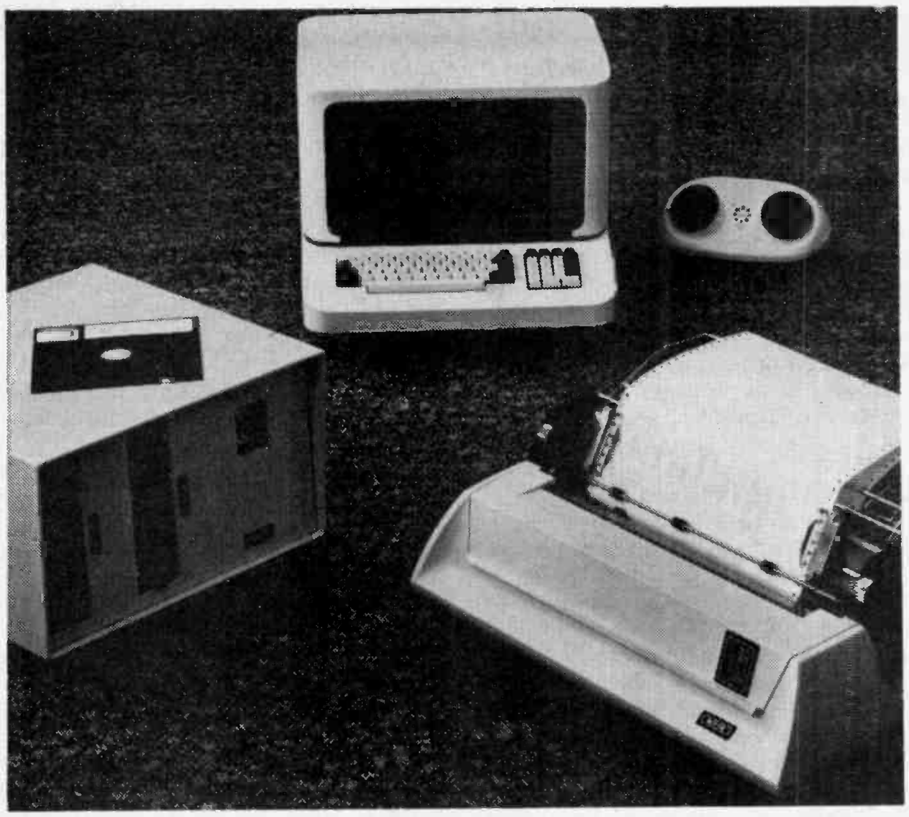


Fig. 5. Here's what the MCM 900 system looks like. This collection costs around $20,000. The keyboard-display unit contains the guts. The display itself is a 12 inch screen with 21 lines by 96 characters.

can tear himself away from the console!

VIDEOBRAIN

First out with APL appears to be Umtech with their F8 based Video Brain. Fitting into 13k - worth of ROM-in-a-cartridge, APL/S is a subset of 'full APL', although there is no standard APL. While a standard for APL is being worked on, big machine versions provide a reference which differs only slightly from one to another, and then only in housekeeping facilities rather than in the basic notation.

Z80 AND VANGUARD

Vanguard Systems Corp. have announced an APL interpreter on floppy disk for Z80, and of course all companies with Z80 based machines are eagerly waiting for it. It is reported to be 27K.

MICROSOFT

Meanwhile, over at Microsoft, much brain work is going into the development of APL interpreters for all kinds of processors, past present and future, ie 8080, Z80 and the 16 bit micros 8086, Z8000, and 68000. Bill Gates, president and APL Product Manager at Microsoft figures fall '79 will be the time his 8080 and Z80 interpreters will be ready with the 16 bit versions in early 1980. Vanguard are hoping to manage the same feat.

IT'S COMING!

Well, it's almost here. It appears that the development of APL on a 16 bit microprocessor will make the first really comfortable implementation of APL for personal, home, business or educational use. Because of this general purpose nature most new APL machines will be easy to use and hence either contain APL on ROM (fool proof) or on disk (most machines will have a disk or two anyhow).

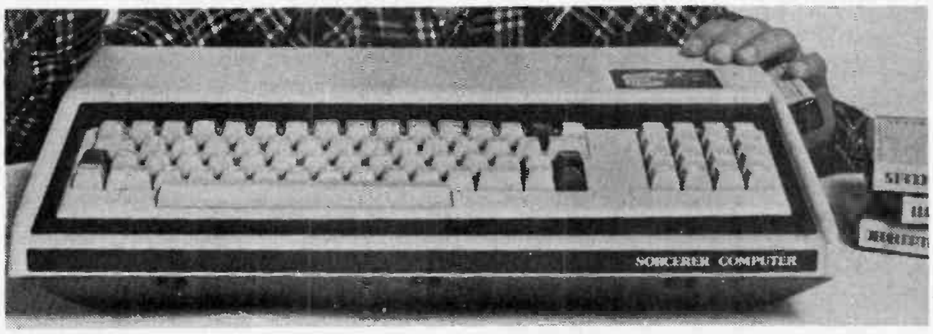


Fig. 6. This is the Exidy Sorcerer, which currently runs BASIC language. Plug in ROM-PACs, just visible to the right, will allow the use of other languages, with APL to be ready in the fall.

Where Is APL Going?

Microsoft's Bill Gates has a bit to say.

Microsoft is a name in the microcomputer software field regarded with a great deal of awe. The US trade magazine 'Electronics' describes Microsoft as a 'small...software house', but with customers like Radio Shack and Commodore (TRS-80 Level II and PET BASICs) who are the leaders in consumerized home computing, and a huge list of other microcomputer companies, Microsoft has to be *the* major 'force' in micro software.

Bill Gates is the president of the company, and the APL job is largely his baby. So we asked him for his views on the micro APL scene.

NEAR FUTURE

What is about to happen in the way of micro APL?

'APL will see an incredible increase in popularity because it will be exposed to so many new people. To date, it has been an expensive language to use and is almost never introduced to first time computer users. APL's strengths assure that a significant percentage of personal computer users will adopt it as "the language". However it is not the ideal first time language and has some limitations of its own. BASIC will continue to dominate in this role, although specialized languages will be supported by personal computer manufacturers. Microsoft will introduce APL on the TRS-80, Exidy Sorcerer, Interact One, Nascom and NEC TK-80 in 1979.

Also the old time APL users will enjoy the low cost and ease of access to the low priced machines. These old time users will be key in generating new converts. New tutorial material making APL seem less mathematical and not forcing the user to see the utility of all the operators will be required. The statistical, accounting, and model type applications that APL is excellent at will be key ones for the new small business computers. With APL's following, IBM's support of it and its strength, our OEMs have decided they can't ignore it. Our response on the product to date has been incredible.'

FIRST VERSION

Details of the 8080/Z80 version:

'Our 8080/Z80 version (long awaited) will be out in April running under CP/M. Other versions such as the virtual workspace one will follow.... Equivalence with the 5100 was my goal — and I met that with 24K of code.

We will license our APL to most of our existing OEMs.'

PERFORMANCE

What kind of performance is expected from micro APL?

'Microsoft APL has all the features of IBM APL/SV (360/370) APL except that the I/O and shared variables are handled differently and the math will not be totally the same since we use binary exponents where IBM uses hex. Our APL is twice as fast as the 5100 which is very adequate and often better than remote APL timesharing systems. Our 8086 and Z-8000 APLs will be at least five times faster and will be available in early 1980. This APL will outperform APL on most minis.

The string search and block transfer are an advantage that our Z-80 version will use. In specific cases this could cause a 2:1 speed difference, but overall the Z-80 to 8080 difference will be less than 15% at equal clock speeds. More important are the design techniques which allow a 32K byte work space to be equivalent to almost twice that much workspace on the 5100 through more efficient management and special representations.'

FAR FUTURE

Looking forward about a year Gates sees some of the following features being offered:

Workspaces upto 40K with virtual workspace to follow.

Libraries on disk.

Unlimited number of dimensions for arrays.

16 digit computation (floating point)

Availability of all primitives normally found.

Sophisticated system commands, error messages.

Initially only SAVE and LOAD workspace 'file' operations, but later random access I/O on arbitrary objects.

Later on Gates thinks 16 bit APL will have up to 10 meg workspaces, same library facilities as I.P. Sharp. Significantly, he comments that many language extensions may be persued, something he feels IBM has stifled. 'APL should not be allowed to stagnate. It is very weak in control structures. General lists should be added.'

AN EXCITING NEW COMPANY

Starting a new company can be a thrill, and it sounds like Microsoft is an example:

'Paul Allen and I started Microsoft in November 1974. The incredible potential for the microprocessor when combined with good software presented a unique opportunity for us to start on our own. The conventional languages and our enhancements of them (BASIC, FORTRAN and COBOL) have kept us extremely busy and delayed our introduction of APL for over 18 months. Providing these packages for the 16 bit processors as well as our new 8-bit products: a BASIC compiler, a Data base manager, a PASCAL based systems programming language and of course APL, will be our primary emphasis in 1979.

The APL project was started in January 1976. At one time people had doubted the ability of microprocessors to support high level languages. After BASIC had disproven this, APL seemed like a great follow-on product. APL provided a new challenge because of its complexity and requirement for lots of memory. However, FORTRAN got higher priority and only one person was left on APL after November 1976. FORTRAN was introduced in June 1977. COBOL was introduced in June 1978.

Microsoft now employs 14 full time technical people and has moved to new and bigger offices in Seattle, Washington, as well as purchasing a DEC 2020 for in house development.'

THE MCM STORY

This company, based in Toronto and Kingston Ontaio, has had a very interesting past, and a unique part in the development of APL machines.

In late 1971, early 72, two fellows by the names of Mers Kutt and Gord Ramer started the company. Their first product was the MCM 70 desktop APL computer, which was based on the 8008 the first 8 bit microprocessor. This model was quickly superceded by the MCM 700 which was a slightly modified version. This machine had dual cassette drives, and one line alphanumeric display. 32K of ROM contained the interpreter, with 2K RAM provided, expandable to 8K.The 700 even came with battery power, enough for the APL addict to get his fix anywhere! This machine was offered for about $8000 in 1974.

Kutt left the company in mid 74, and in the spring of 75 Ted Berg became it's president.

In mid 76 the company's second model, the MCM 800 was introduced. Because the APL interpreter already developed was so good, but the 8008 a little slow, the 800 was centred on a CPU board built with a couple of ALUs and other supporting chips imitating the 8008. This resulted in a computation power improvement of around 10 over the 700. The 800 is generally used in a system with dual floppy disks, with perhaps a printer. About $9,000 could get you the 800 itself, which incorporated a VDU and single cassette, and came with 8K RAM.

In mid '78, Berg moved to take on MCM's US distributor Interactive Computer Systems Inc. The company's new president, Chuck Williams explains that with MCM's third generation machine, the MCM 900, the marketing strategy can now be more solidly aimed at a wide market of business applications. The 900 is based again on an 'imitation' high speed 8008, using a pair of 2901 4 bit slice processors, giving an increased performance of 5—10 over the 800 according to John Koiste, General Manager of MCM's Kingston R & D and production facility. Williams likens this improvement from 800 to 900 to the production of automobiles with top speeds of 60 and 80 miles an hour. While the performance improvement is relatively small, and the first car will accomplish transportation, there is a certain threshold level at which the product becomes widely acceptable. This, Williams feels, has happened with their 900, which strongly competes with timesharing and other more expensive models. The 900 itself is priced around $10,000, but is really intended for use with a dual disk and printer. Essentially it is part of an $18 — 20,000 business problem solving package.

As for the future, Koiste admits that MCM has gone about as far as it can with using the 8008 interpreter and will have to develop new software around 'more up-to-date technology'. This we take to mean they are looking at 16 bit microprocessors. Meanwhile, Williams says he isn't worried about competition from the hobby computer manufacturers, even as they pursue APL on 8 and 16 bit machines. He feels his personnel and their experience are the company's strengths, and that they will stay ahead of the game in the business market, where they have seven more years of experience than the newcomers.

I. P. SHARP

An interesting story came to our attention recently in connection with micro APL. I.P. Sharp Associates, known worldwide for their APL communications network and software expertise, were apparently working on a micro based APL consumer product! Sharp's Steve Kohalmi described the project. A major Canadian manufacturer of entertainment products went to Sharp with the idea, and developed the hardware prototype while Sharp worked on the software. The product was based on a GI 1600 microprocessor, and was to have floppy disk(s) and some colour graphics capability. The APL interpreter, which was developed on a PDP11/34 simulating GI code, has dynamic workspace management, floating point numbers, but only one dimensional arrays. 16K of 16 bit words was required. Unfortunately the whole combination was never tested as interest in further development has appeared to lapse. Anybody want to take over an almost debugged GI 1600 APL? Call Sharp.

ACKNOWLEDGEMENTS

Our thanks to the following people who were very helpful in the preparation of this article:

Bill Gates, Microsoft;
Chuck Williams, John Koiste, MCM Computers;
Steve Kohalmi, I. P. Sharp Associates;
I. P. Sharp Newsletter

SUGGESTED BOOKS

Kenneth E. Iverson: "Elementary Analysis"; APL Press, 1976.

The APL Press has available a comprehensive collection of publications on APL, the interested reader should write for a complete list.

Leonard Gilman and Allen J. Rose: "APL An Interactive Approach"; John Wiley and Sons Inc, 1974.

A. D. Falkoff, and K. E. Iverson, "APL Language", IBM Corp, 1975 (Form No. GC26-3847)

ADDRESSES

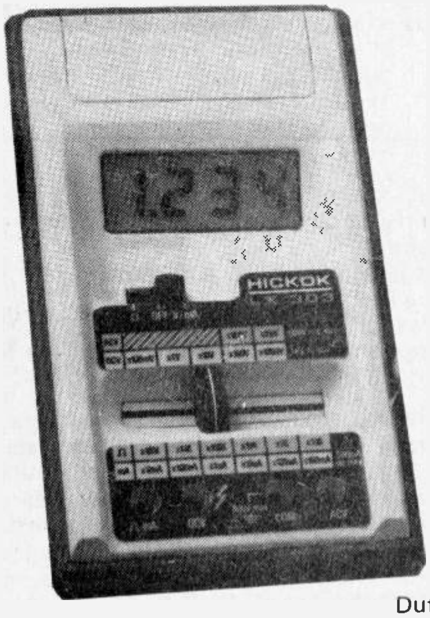
APL Press, Box 378, Pleasantville NY 10570.

Exidy Inc., 969 West Maude Ave., Sunnyvale, CA 94086.

MCM Computers Limited, 6700 Finch Ave. West, Suite 600, Rexdale, Ont., M9W 5P5.

NEW LOW COST LX303 DMM

$99.95



Operates up to 200 hours on a single 9 volt battery. 100 mV DC f.s. sensitivity. Features auto zero, polarity, overrange indication. Nineteen ranges and functions. With easy-to-read ½" LCD display for convenient use in any kind of light. Removeable cover stores test lead set furnished as part of the unit. Excellent overload protection. Colour coded panel. Carries a full one year warranty. Available accessories include 10 kV probe/adapter, 40 kV probe, 10 amp DC shunt, AC adapter and deluxe padded vinyl carrying case.

SPECIFICATIONS:

DC VOLTS (5 RANGES): 0.1 mV to 1000V; Accuracy ±0.5% rdg ±0.5% f.s.; Input imped: 10M ohm; Max. input 1kV except 500V on 200mV range. AC VOLTS (40Hz to 5kHz): 0.1V to 600V; Accuracy: ±1.0% rdg ±0.5%f.s. (−2db max. at 5kHz); Max. input: 600V.

RESISTANCE (6 LOW POWER RANGES): 0.1 ohm to 20M ohm; Accuracy: ±0.5% rdg ±0.5% f.s. (±1.5% rdg on 20M ohm range); input protected to 120 VAC all ranges.

DC CURRENT (6 RANGES): .01nA to 100mA; Accuracy: ±1.0% rdg ±0.5% f.s.

DIMENSIONS AND WEIGHT: 5-7/8" x 3-3/8" x 1-3/4", 8 oz.; POWER: 9V battery (not included) or Hickok AC adapter; READ RATE: 3/sec.

HICKOK
the value innovator

Duty and FST included.
Chargex (Visa) accepted.
Ont. res. add 7% sales tax
F.O.B. Ajax.

ROGERS electronic instruments ltd.
P.O. Box 310, 595 Mackenzie Avenue Units 1 & 2.
Ajax, Ontario L1S 3C5. Tel. (416) 683-4211