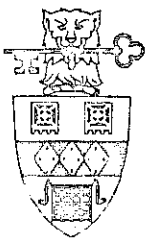
VECTOR

- J-APL86 – latest news
- Future direction of APL.
- APL2 idioms
- More from APL86.
- News, reviews and APL product guide.



The Journal of the British APL Association

A Specialist Group of the British Computer Society

Vol.3 No.3 January 1987

Contributions

All contributions to VECTOR should be sent to the Editor at the address given on the inside back cover. Letters and articles are welcomed on any topic of interest to the APL community. These do not need to be limited to APL themes nor must they be supportive of the language. Articles should be submitted in duplicate and accompanied by as much visual material as possible, including a photograph of the author. Unless otherwise specified each item will be considered for publication as a personal statement by its author, who accepts legal responsibility that its publication is not restricted by copyright. Authors are requested wherever possible to supply copy in machine-readable form ideally text files on a 5¼" IBM-PC compatible diskette. For other standards, please contact the Editor beforehand. Program listings should indicate the computer system on which they have been run. APL symbols should be displayed on a separate line and not embedded in narrative . Except where indicated, items published in VECTOR may be freely reprinted with appropriate acknowledgement.

Membership Rates 1986-87

Category	Fee p.a. £	Fee p.a. $	VECTOR copies	Passes
Nonvoting student membership	5		1	1
UK Private membership	10		1	1
Overseas private membership	18	27	1	1
Supplement for airmail (not needed for Europe)	8	12		
Corporate membership (UK)	85		10	5
Corporate membership (Overseas)	140	210		
Sustaining membership	360		neg	5

The membership year runs from 1st May to 30th April. Applications for membership should be made on the form at the end of the journal. Passes are required for entry to some Association events and for voting at Annual General Meetings. Applications for student membership will be accepted on a recommendation from a course supervisor. Overseas membership rates cover VECTOR surface postage and must be paid in £UK.

Corporate membership is offered to organisations where APL is in professional use. Corporate members receive multiple copies of VECTOR and are offered group attendance of Association meetings. Partaking individuals need not be identified but a contact person should be nominated for all communications.

Sustaining membership is offered to companies trading in APL products; this is seen as a method of promoting the growth of APL interest and activity. As well as receiving public acknowledgement for their sponsorship, sustaining members receive bulk copies of VECTOR, and are offered news listings in the editorial section of the journal and opportunities to inform APL users of their products via seminars and articles.

Advertising

Advertisements in VECTOR should be submitted in typeset camera-ready A5 portrait format with a 20 mm blank border. Illustrations should be black-and-white photographs or line drawings. Rates are £250 per page. A6 and A7 sizes are offered subject to layout constraints.

Deadlines for advertisement bookings and receipt of camera-ready copy are given beneath the Quick-Reference Diary.

Advertisements should be booked with and sent to Steve Lyus, whose address is given beneath the Index of Advertisers.

CONTENTS

Info Center/1

an IBM licensed program that helps business professionals perform their daily tasks quickly and productively

Info Center/1 provides an integrated, multifunction information center environment compatible with predecessor products such as ADRS II and APLDI II. A full-screen interface, with prompts and extensive help facility, provides easy access to the following powerful general business functions, as well as providing the full power of APL:

Query System

The Query System provides a simple, effective way to interactively access, analyze, manipulate, and report information stored in files of up to several hundred megabytes.

Reporting System

Provides an organization with a single, comprehensive system for generating and maintaining reports. Standard calculations can be defined and stored for future use. Calculations can be made with predefined functions and with APL.

Data Entry and Validation

This tool allows information center personnel to tailor panels for users to display, update, and enter data in column format.

Financial Planning System

The Financial Planning System provides a set of 60 modeling routines that work with the Reporting System and address periodic data. Some examples are:

- Financial analyses and plans
- Statistical analyses and projections
- What if analyses and modeling
- Project evaluations and risk analyses.

Business Graphics

The Business Graphics facility is a particularly powerful yet flexible tool for interactively producing the following types of charts: line graphs, surface charts, histograms, pie charts, scatter plots, bar charts, stacked bar charts.

Technical Data

Info Center/1 is an IBM Licensed Program, Program Number 5668-897.
The program runs under CMS and TSO together with the following IBM programs or their equivalents: APL2 or VS APL, Application Prototype Environment, GDDM (Graphical Data Display Manager).
Some examples of terminals supported are: IBM 3277, 3279, 3270 PC/G and GX.

Editorial: "Train up a child ..."

by David Preedy

This issue of VECTOR unashamedly devotes several pages to various aspects of the I-APL project, of which I hope all our readers are by now at least aware. Although I-APL is a separate activity from any other organisation, it has already at the time of writing in early-November received backing from the British APL Association and from SIGAPL. The transatlantic nature of its support is reflected not only in the make-up of the I-APL committee, but also of those others involved in the development of the interpreter itself and of the infrastructure to ensure its success – the documentation, supporting material and the "business plan"; spawned in Manchester, I-APL should be developed by Dallas.

My direct knowledge of education is limited to my own experiences and to those of my close family and in particular my own children. As a mathematician, I cannot fail to see how much more appropriate an APL background would have been to my own education, than was the mixture of Algol and Fortran that constituted the norm before the days of micros and BASIC. However my views of the computing needs of today's school-children are coloured more by what I have seen as an outsider looking in on my children's education.

In general the teaching profession has shown considerable inventiveness and imagination in taking the microcomputer to heart. Of course there have been cases where the computer has been used trivially to mimic the teaching of arithmetic by rote, and of course it would be better if all schools had the resources to buy more computers. But equally there are many dedicated teachers who recognise the computer as a totally new type of teaching aid, that should not be used merely as a glorified adding machine. Also there is much innovative software available for the educational market – the Mary Rose project and the work done with LOGO would seem to be classic examples.

Above all though we must avoid the danger that all the school-children interested in computers become so accustomed to the BASIC approach to programming that they regard its looping, scalar techniques as sacrosanct. (It was Anthony Camacho, I think, at APL86 who expressed the fear that we are training people to be unable to use the benefits of parallel computers.) APL seems to have three strong advantages in the educational area. Structurally, it reflects the mathematical concepts being taught much more clearly than any other language; it is after all a sophisticated mathematical notation in its own right. As a calculating machine, it provides a powerful tool for pupils to use to support other work they may be doing – analysing experimental results, and so on. And for the teacher, its brevity should help in preparing functions, where appropriate copied from written material, to illustrate the work in hand. The speed with which results can be obtained with only a modicum of APL expertise must be a key factor in retaining the interest of youngsters whose attentiveness fades when faced with a daunting stream of PRINT$ IF THEN ELSEs.

What then can we as APL users do to assist the momentum of the I-APL band-wagon? The key factor is communication. The small core of educationalists in the APL community cannot expect personally to reach a large number of teachers; they can however attempt to concentrate their own resources on the key decision-makers – LEA Mathematics Advisers and the like. What the rest of us should be able to do is to contact those teachers we know personally, especially those with an interest in mathematics and computers, and tell them about I-APL. However we must give them a two-fold message – not only that a free APL interpreter is available, along with suitable supporting material, but also why it should be important to them.

Now we can all explain in our own way why APL helps us, but the niceties of component file systems and nested arrays may well be irrelevant to the class's uses of computers. What we need to do between now and the start of the publicity drive in the Summer is to encourage those teachers already using APL to tell us lay-people what particular characteristics of APL are most relevant in the educational context. Then we have a real prospect of introducing APL effectively to a whole new generation.

> "Train up a child in the way he should go: and when he is old, he will not depart from it." Proverbs, xxii, 6

Dates for future issues of VECTOR

	Vol 3 No 4	Vol 4 No 1	Vol 4 No 2	Vol 4 No 3
Copy date	30 Jan 87	24 Apr 87	24 Jul 87	16 Oct 87
Ad. booking	20 Feb 87	22 May 87	21 Aug 87	13 Nov 87
Ad. copy	27 Feb 87	29 May 87	28 Aug 87	20 Nov 87
Distribution	April 87	July 87	October 87	January 88

Quick-reference diary

compiled by David Preedy

Date	Venue	Event
1987		
6-7 January	London	APL in Engineering & Information Science Organised by South Bank Polytechnic
20 February	London	British APL Association meeting
20 March	London	British APL Association meeting
11-15 May	Dallas	APL 87 – APL in transition
6 June	London	British APL Association AGM & meeting
18 September	London	British APL Association meeting
16 October	London	British APL Association meeting
20 November	London	British APL Association meeting
1988		
15 January	London	British APL Association meeting
18 March	London	British APL Association meeting
20 May	London	British APL Association AGM & meeting
16 September	London	British APL Association meeting
21 October	London	British APL Association meeting
18 November	London	British APL Association meeting

All British APL Association meetings are to be held at the Royal Over-Seas League, Park Place, near Green Park tube station and start at 2pm.

Please note the date of the AGM, which has been moved to June 6th to avoid a clash with APL-87 in Dallas!

MERCIA Software Limited

WSFULL problems with APL*PLUS/PC?

Before	After
WSSIZE	WSSIZE
487568	747408

Could you use those extra Bytes? – give us a call to find out how.

Mercia also offer a full range of consultancy and education services – make a note in your diary of our 1987 course schedule:-

Introduction to APL*PLUS/PC – 3 Day Course £350
(In House £1200)

January 20, 21, 22
March 17, 18, 19
May 19, 20, 21

APL*PLUS/PC Enhancements – 2 Day Course £240
(In House £800)

February 17, 18
April 1, 2
May 5, 6

System Design with APL*PLUS – 3 day Course £375
(In House £1200)

January 27, 28, 29
March 24, 25, 26
May 12, 13, 14

MERCIA SOFTWARE LIMITED

Aston Science Park, Love Lane, Birmingham B7 4BJ.
Telephone: 021-359 5096

APL course diary

Many of the APL vendors included in the VECTOR APL Product Guide offer courses in APL and related topics. For a full list readers are recommended to look under the relevant section of the product guide. This section gives course dates for those suppliers who have prepared their course schedule at the time of going to print.

January 1987

14	Beginners APL	MicroAPL
20-22	APL Fundamentals	Cocking & Drury
20-22	APL*PLUS/PC Introduction	Mercia Software
21	Intermediate APL	MicroAPL
26-29	APL*PLUS PC Intermediate	Cocking & Drury
27-29	System design with APL*PLUS	Mercia Software

February 1987

3-4	Statgraphics	Cocking & Drury
10-12	APL Fundamentals	Cocking & Drury
11	Advanced APL	MicroAPL
16-19	APL System Design	Cocking & Drury
17-18	APL*PLUS/PC Enhancements	Mercia Software
18	Intermediate APL	MicroAPL
24-26	APL Fundamentals	Cocking & Drury

March 1987

10-12	APL Fundamentals	Cocking & Drury
16-19	APL*PLUS PC Intermediate	Cocking & Drury
17-19	APL*PLUS/PC Introduction	Mercia Software
24-26	APL Fundamentals	Cocking & Drury
24-26	System design with APL*PLUS	Mercia Software
31-¼	Statgraphics	Cocking & Drury

April 1987

7-9	APL Fundamentals	Cocking & Drury
21-22	APL*PLUS/PC Enhancements	Mercia Software
28-30	APL Fundamentals	Cocking & Drury

May 1987

5-6	APL*PLUS/PC Enhancements	Mercia Software
11-14	APL System Design	Cocking & Drury
12-14	System design with APL*PLUS	Mercia Software
19-21	APL Fundamentals	Cocking & Drury
19-21	APL*PLUS/PC Introduction	Mercia Software
27-28	Statgraphics	Cocking & Drury

APL CONSULTANTS

LONDON & READING

Account Managers	(6 years+)	to	**25K**
Senior Consultants	(4-6 years)	to	**21K**
Consultants	(2-4 years)	to	**17K**
Junior Consultants	(1-2 years)	to	**13K**

Are your APL skills and potential being recognised and rewarded?

Cocking & Drury consultants have been implementing successful decision support applications for 10 years, with clients who appreciate the productivity benefits of APL.

In our professional team you will experience a range of APL environments - APL*Plus, VSAPL, APL2 and Unix, on both mainframes and micros. You will also be developing systems which, increasingly, need to interface with non-APL Information Centre products.

Of course as the leading APL consultancy, in a rapidly expanding market, we offer a rewarding career with first class benefits - profit sharing, free health insurance, and a non-contributory pension.

For further details call Ralph Wilson on 0734 588835

COCKING & DRURY LTD.

155 Friar Street, Reading, RG1 1HE

General Correspondence

The VECTOR working group welcomes correspondence on any topic affecting the APL community. All such letters should be addressed to the Editor and should indicate whether they are intended for the general or technical section. Letters containing APL code will normally appear in the Technical section of VECTOR, and authors are asked to observe the requirements on the inclusion of APL code stated on the inside cover. The Editor reserves the right to edit any letter unless the writer states that the letter is to be published in full or not at all.

APL standards

From Mr Paul Barnetson 21st October 1986

Sir,

At a recent meeting of the BSI APL Panel, there was a long discussion on how we could best attract new members to join us in our exciting work. I hope that an appeal to the VECTOR readers may result in some volunteers coming forward.

Most of us have moaned because a package that runs on one computer doesn't run on another. Result: you have to rewrite it!

This problem is now being overcome by the APL Standard. The document is shortly to be published as an ISO (International Standards Organisation) Standard, and this will be followed soon after by the same document appearing as a BSI (British Standards Institute) Standard.

But . . . it won't cover all the modern facilities of current APL implementations. Both the ISO APL sub-committee and the BSI APL panel are gearing up to work on standardising some of these modern facilities:

> generalised arrays, operator extensions, scope of names, file input/output, complex arithmetic, exception handling, etc.

I would like to ask your readers three questions:

> Do YOU use these techniques in APL?
>
> Do YOU want to see them standardised?
>
> Are YOU prepared to work in this area?

If the answer to any of these is "Yes", then you'll want to participate in the future work of the BSI APL panel. We meet once a quarter, on average, and normally in the London area. There will also be opportunities for members to attend the international meetings of the ISO APL sub-committee.

Further information can be obtained from either of the following:

David Ziemann, BSI APL panel chairman, Cocking & Drury, 01-493 6172

Paul Barnetson, BSI APL panel secretary, IBM, 0705-323054

Yours sincerely,

Paul Barnetson,
Secretary, BSI APL panel,
IBM United Kingdom Ltd.,
P.O.Box 41,
Northern Road, Portsmouth,
Hampshire. PO6 3AU.

(Editor: I am sure we all wish you luck in attracting volunteers for this interesting and important challenge.)

Comparing Languages

From Mr Peter Branson 20th October 1986

Sir,

I have used APL for many years (mainframe) but had to leave it for a while. Coming back to it, I joined the B.A.A. several months ago, and back issues of VECTOR have been most helpful in highlighting recent developments. I enjoy my copies of VECTOR so please keep up the good work.

A book review is enclosed, for the "Handbook and Guide for Comparing Computer Languages". You will see that I came across it by chance in a local library; Felicity at Mine of Information says it is not on their lists, and she hasn't been able to find a U.K. supplier yet for me.

I was surprised to find, when talking to people at the Royal Over-Seas League last week, that no-one appeared to have heard of this book. Since I had already made detailed notes, it seemed sensible to make them more widely available to the VECTOR readers.

Yours sincerely,

Peter Branson,
Oaklands Cottage, Wray Common,
Reigate, Surrey.

(Editor: Peter's review is included in the reviews section later in this issue of VECTOR. Perhaps any reader knowing of a supply of the Handbook could let us know.)

I-APL

From Mr Simon Barker,

Sir,

Please find enclosed a donation towards the I-APL appeal. It's pleasing to see something so positive and (hopefully) far-reaching being attempted.

My own feelings about the objectives of the I-APL project are rather mixed; partly because it seems that the project is concerned with making APL the new BASIC (although it can certainly live up to being a Beginner's All-purpose Symbolic Instruction Code); and partly because I feel that a subset of plain old "vanilla" APL will have a hard time competing against full-scale implementations of Fortran, Pascal and some advanced BASICs which are already in wide use.

The area where APL really needs credibility is in the business arena because it is here that a language can grow in stature enough to be taken seriously elsewhere. For this reason, in parallel with the I-APL project, I think there should be a drive to develop powerful, cheap software written in APL to exploit the new icon-driven machines that are filling today's computer market. Just imagine how popular APL might have been if, say, Lotus 1-2-3 had been written in it.

To this end I salute MicroAPL for taking the initiative and producing APL for the Macintosh, Atari ST and the Amiga, along with all-important run-time versions of the interpreter.

If software developers have got any sense, they will soon realise how easy it is to produce well-specified, full-featured software, quickly, easily and cheaply using APL. The I-APL publicity drive might well help towards this end.

Finally, may I say how much I enjoyed APL86 and how superbly VECTOR Vol 3, No 2 captured the excitement and sheer pleasure of that event.

Yours,

Simon Barker,
55 Conisborough Crescent,
Catford, London SE6 2SP.

(Editor: Your contribution to the I-APL project has been forwarded to the I-APL committee. Readers might like to note that I-APL is separate from the BAA and that I-APL matters should normally be addressed to the I-APL committee, specifically Anthony Camacho, David Ziemann or Norman Thomson – in the U.K.

Thank you for your comments on last VECTOR; we like to receive comments from readers – even unfavourable ones which give us useful feedback to improve your journal.)

APL.68000 FOR THE COMMODORE AMIGA

APL.68000 on the Amiga offers a uniquely friendly environment in which to program in APL. It offers the simplest method of writing applications which take full advantage of the Amiga features, allowing systems written in APL to set a new standard of professionalism and integration with other Amiga software.

FEATURES:

- **Uses standard Amiga user interface**
- **Built-in full screen function editor**
- **Access to Amiga Dos native files**
- **Built-in VT100 APL/ASCII terminal amulation**
- **Full clipboard support for data exchange**
- **APL can be started from workbench**
- **Common system commands as pull-down menus**
- **Full printer support**
- **Runs in its own window**
- **User-defined pull-down menus**
- **User-defined Dialog and Alert boxes**
- **Full interface to Amiga graphics facilities**
- **Arbitrary I/O via serial ports**
- **Session manager allows editing of screen lines**
- **Workspaces can be set as run-time applications**
- **Applications can use standard ASCII keyboard**
- **Applications can detect mouse position and state**

Price: £200 excl. VAT

MICRO APL

MicroAPL Limited
Commercial Offices: 19 Catherine Place, London SW1E 6DX
Telephone: 01-834 9022

British APL Association News

BAA Committee News

by Dick Bowman

There are two items of interest arising from recent Committee meetings, both aimed at improved communications and information exchange in the global APL community.

Sharp Mailbox code.

I P Sharp Associates are Sustaining Members of the BAA and their cooperation during the organising stages of APL86 was most helpful; in particular the access which they provided to their Mailbox system.

They have made the most welcome gesture of allowing the BAA to continue using the Mailbox, which means that two things can now happen:

We have created a Mailbox group called <FBAA> (Friends of the British APL Association) which can act as an address for any communications of interest to members of the BAA. Join this group if you want to receive any of this news etc.

Also you can send messages direct to the BAA Committee. At present we have no group specifically for this purpose (one day the Secretary will doubtless get round to it) and you should address your message to either <BOW>, <MEL>, or <ZIEM>.

International Journal Exchange.

There was a somewhat inconclusive meeting of many of the international APL 'clubs' at APL86 – one thing which we were able to achieve being that we wanted to revitalise the longstanding but lately moribund exchange of journals between the groups.

We have contacted all of the groups that we know of and now have journal exchange agreements with the Swiss, German, Bay Area and Australian Groups. You should begin to see some summarisation of the contents of the various journals in forthcoming VECTORs; what you won't be seeing is lengthy extracts from their articles.

What we want to achieve is to let all of our membership know what's happening in APL worldwide and we'd particularly encourage you to join the individual groups if you think their activity looks interesting.

If, by any chance, there are any other groups publishing APL journals who would like to participate in this scheme please contact a member of the BAA committee as soon as you can.

News from Sustaining members

APL People

By the time you read this the summer holidays will have long gone and we will all just be recovering from Christmas – ready for another busy APL year ahead.

APL People's associated enterprises continue to prosper. APL Tran-Plan is progressing with its work on several transportation studies in North America; H Walton Technical Services are busier than ever with PEFAC – the computerised estimating system; and APL People's consultancy business is steadily growing. Recently their employment agency has been engaged by clients in the U.S.A. to fill vacancies in New York, and potential business in North America looks most promising.

Since APL86, where APL Software Limited got under way, the company has contracted to market a range of software for both mainframe and PC environments (see Product Guide). Other software is being evaluated and should become available in 1987. Initial interest has been encouraging not only in the current portfolio of software, but also from companies wanting their in-house systems evaluated with a view to marketing them. How much "new" application software might be discovered for the benefit of the APL community.

APL Software Technology (UK) Ltd

APL Software Technology's investment into APL Software Limited (together with APL People) has begun to show a healthy level of interest. The new company will be a powerful arm to the marketing and sales of the products they offer.

The latest release of Powertools has been well received by their client base, the extended facilities giving a boost to the development of application systems. New versions of this release have now been made for the MicroVax, under Unix, and the Wang PC.

The mainframe Relational Database System RDS is now available on the IBM PC and is currently undergoing beta-testing with one of their clients. There has been much interest in RDS, the many enquirers including a US banking organisation.

APL Software Technology looks forward to a busy and interesting 1987.

Cocking and Drury Ltd.

Cocking and Drury have recently signed an agreement with Uniware, the leading French APL software house. The deal gives Cocking and Drury an exclusive dealership throughout Great Britain and Ireland for Unitab, The APL Debugger and a Statgraphics add-on module.

Unitab is a PC workspace database manager that lets you manipulate and browse APL data in a spreadsheet-like manner. The system is window-oriented and uses pop-up menus to interact with the user. Although the product has been available to French-speaking users for some time, the latest version includes English translation for all help, prompts and documentation. The software contains hooks to allow programmers to customise the package with their own code.

The APL Debugger is a development and debugging aid for APL*PLUS PC application writers. It lets you easily step through an executing function, producing a scrollable full-screen display of the current expression and its result. Local variables may also be examined and modified.

The Statgraphics add-on module for Correspondence Analysis is the first of a series of third-party add-ons that will be made available for use with Release 2 of Statgraphics. Cocking and Drury welcome approaches from users who have developed software that could be marketed in this way.

Release 2 of Statgraphics was announced in September last year. As well as general speed-ups throughout the system, the new release boasts enhanced data management features, direct dBase, Lotus and Symphony import/export, a variety of new statistical procedures, multiple plot overlaying and a plot 'zoom' option. In addition, a mechanism allowing APL developers to customise the package by adding their own menu options is provided.

Release 6 of APL*PLUS PC is selling well. New features include arrays without imposed size limits, support for the exciting HP LaserJet + printer, full graphics character set support (no APL ROM needed), improved file system facilities, and enhancements to the ⎕WIN full-screen system function. The file system improvements permit dynamic access to DOS directories via APL library numbers, and generalised native file access via complete DOS path names. The documentation is 'all new' and includes an invaluable spiral-bound pocket reference guide.

Cocking and Drury are proud to announce their first mainframe compiler trial at a UK customer. In a study for another client, their expected savings for using the APL*PLUS Enhancements and Sharefile component filing system (rather than APL2) amounted to £100,000 over a five-year period. Savings are expected to be even larger when the compiler is installed. Cocking and Drury is also now offering compiler training seminars throughout Europe.

Cocking and Drury and STSC have recently signed a dealership agreement for Dataport, the spreadsheet system for mainframe based VS APL users, and Cocking and Drury now support all existing Dataport users in Great Britain and Ireland.

On the education front, bookings are being taken for two new public courses; the four-day APL*PLUS PC Intermediate course (plenty of practical tools and systems design) and a two-day course for Statgraphics users. A new one-day in-house nested arrays course is also newly available anywhere in Europe.

Dyadic Systems Limited

Dyadic Systems has completed its transition from a software-only company to a supplier of complete APL systems. In July, Dyadic became one of the first Authorised Dealers for the exciting new IBM 6150 Microcomputer, and has since announced an Altos dealership and an arrangement with Sun Microsystems. As a result, Dyadic is able to offer and support a comprehensive range of APL systems to meet a variety of business needs and budgets.

At the bottom of the range is the Dyalog APL Coprocessor, a plug-in board for the IBM PC. The system has an NS32000 processor, hardware floating-point, up to 4Mb RAM and 16Mb of virtual address space. The Dyalog APL Coprocessor provides a concurrent DOS and multi-user APL/Unix environment in an IBM PC or compatible.

Mid-range systems are based on the IBM 6150 and Altos 3068 computers. The IBM 6150 has a 32-bit CPU based on Reduced Instruction Set (RISC) technology. Its operating system is AIX, an enhanced version of Unix V.2 with an improved user-interface, and it has a PC AT coprocessor for DOS applications. Communications support Ethernet, Token Ring and SNA. The 6150 is seen as a strategic system for IBM, and is expected to double in power and capacity annually. The system includes a special version of Dyalog APL which supports interfaces to IBM 6150 SQL and to the Advanced Graphics Support Library (GSL). A typical configuration includes 4Mb RAM, Floating-point processor, 140Mb disk, a high-function bit-mapped console, several IBM 3163 ASCII terminals and/or high quality Lynwood APL/Graphics VDUs, and an IBM 4201 Proprinter. All devices are supplied complete with APL character support.

The Altos 3068 is a 68020-based alternative. Its special features include a local-area network which supports IBM PCs and compatibles. This provides file transfer, central storage and sharing of DOS files, shared printers and the use of the PCs as Dyalog APL terminals to the Altos 3068 host. A typical configuration includes the network, 4Mb RAM, 170Mb disk, diskette, cartridge tape and several high-quality Lynwood APL/Graphics VDUs.

Dyadic also supplies a range of Sun workstations and multi-user computers which can be configured either as a network or as a traditional time-sharing system. Sun workstations have 19-inch high-resolution screen with advanced windowing features. At the top of the range is the outstanding Sun 3/200. This has a 25MHz 68020 CPU, 68881 floating-point co-processor and cache memory. These features alone make it twice as fast as other 68020-based APL systems on the market. Add Sun's optional Floating-Point Accelerator, which is 2-3 times faster than a 68881 co-processor, and you have a very powerful APL microcomputer indeed.

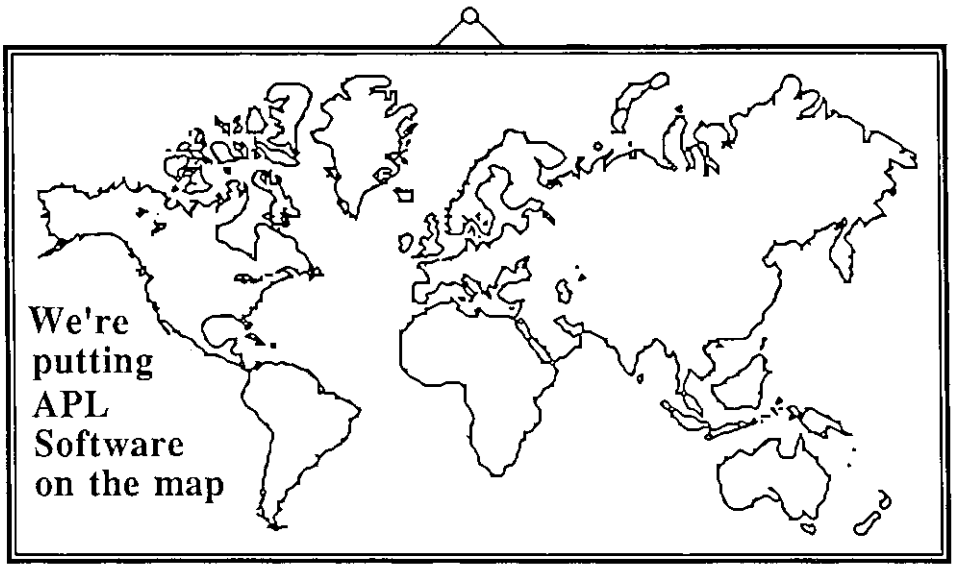
Dyalog APL is not restricted to any particular processor. This means that Dyadic can continue to take full advantage of technical developments in the computer industry to offer competitive high-performance systems for the APL user. Watch for a Compaq 386 implementation early in 1987.

Mercia Software Limited

Mercia Software are looking forward to continuing growth in the APL-related market in 1987. Interest in APL*PLUS/PC is still strong, with their range of programmer productivity aids, such as UNIWARE's Debugger, and STSC's TOOLS and SPREADSHEET MANAGER proving popular amongst the nation's APLers.

On the application front, APL-based packages are making more and more inroads into the business computing market. STATGRAPHICS, helped by a price reduction and a much improved new version, has been very successful, and EXEC*U*STAT looks like bringing the power of APL to a wide range of users (even if they don't realise it). At the time of writing, Mercia are about to embark on a major campaign to bring this excellent product to the attention of the PC user world in general.

The future looks even more exciting with the forthcoming lunch in Spring '87 of a major new APL*PLUS-based system, for which Mercia has been liaising with the Forecasting and Materials Management guru, Professor R.G.Brown. O.R. and logistics practitioners should watch this space for news of LOGOL.



APL Software Ltd., 27 Downs Way, Epsom, Surrey KT18 5LU (03727 21282)

POWER

Enhancing APL.68000

An intriguing computer for an exciting language—the WS-1 and APL.68000. At last the APL programmer can have portability without sacrificing power or capability. Dodge the queue waiting for time on the mainframe and discover the sudden freedom of being able to try out programs anytime, anywhere.

The APL.68000 interpreter is implemented in 86KB of ROM, running under a multi-user, multi-tasking operating system called BIG. DOS. Speed is the essence of APL programming, and now the WS-1 makes development even faster.

Come and see us at APL '86.

APL.68000 on the WS-1 has attractive enhancements such as a powerful component file system, QUAD. FMT function for alpha report formatting, QUAD. CC function for full-screen control, and extended error trapping facilities.

Bundled with the WS-1 are four workspaces: SYSFNS, APLUTIL, FILEUTIL, and SYSCOM. Each gives access to the WS-1's unique capabilities such as control of the built-in speaker phone, microcassette unit, RTC (real time clock), bit-mapped graphics LCD screen, and optional 3.5-inch floppy disk drives.

Compress these capabilities into a sleek footprint measuring less than 13 inches by 11 inches, and you have the ultimate definition of power.

ampère

FOR DISTRIBUTORSHIP INFORMATION AND PRODUCT DETAILS PLEASE CONTACT:

ampère INCORPORATED Asahi Bldg., 7-5-20 Nishi-Shinjuku, Shinjuku-ku, Tokyo, Japan. *Phone:* 03-365-0825. *Telefax:* 03-365-0999. *Telex:* J33101 AMPERE. IP Sharp Mail Box Code AMP (Group Code APLWS).

International APL News

Journal Exchange

We are now regularly exchanging journals with various overseas APL groups; as a result we are seeing several very interesting documents that haven't come our way before. A good example are the proceedings of the annual seminars organised by the New York SIGAPL under the title "APL as a tool of thought". These have been held each April since 1983, so we now have four hefty volumes of proceedings.

The first seminar in 1983 offered workshops discussing APL as a tool of Mathematics, Science, Social Studies, Natural Language, Music and Computer Science. By 1985, the scope had expanded to include Biology, Calculus, Mathematics, Social Studies, Computer Science, Economics, Graphics, Linear Algebra, Music, Statistics, Artificial Intelligence, Databases, Finance, Gaming & Simulation, Manufacturing, Modelling, System Design and a Teacher's Toolbox.

The full lists of contents are as follows:

APL as a tool of thought – 1983

Binomial Distributions	Linda Alvord
Elementary Algebra	Brooke Allen
Computer Simulation in Science Teaching	Charles Waters
Analyzing Problems of Population Growth and Distribution	Tama Traberman
Computer-Authored Tests and Exercises in Grammar	David Michelson
Musical Grammar	David Steinbrook
APL as a Tool for Teaching Computer Theory	Philip Van Cleave
APL in a Liberal Arts College	Donald B McIntyre
Getting Started in APL	Allen J Rose
An Application in Remedial Math	Cecil Denney
Down to BASIC	Jim Lucas
A Personal View of APL	John McPherson
APL	K W Smillie

APL as a tool of thought II – 1984

Using APL to Teach Concepts in Analysis & Design	Wilbur LePage
Meaningful means: Analysis of Experimental Data	Charles Waters
APL Enhances Mathematics Education	Linda Alvord
Quantitative Reasoning in the Social Studies	Tama Traberman
Polygonal Functions and their use in Teaching Calculus	Don Orth
APL as a Financial Tool	Gary Podorowsky
APL in Linear Algebra	Murray Eisenberg
A Demonstration of MEDCAT	Hagamen, Gardy, Bell & Zatz
Implementation of a Virtual Memory APL Interpreter	Ed Cherlin
Bibliography of APL Publications	NY/SIGAPL
What is APL?	Ed Shaw
Domino – an APL primitive for Matrix Inversion	M A Jenkins
The Art of Question Begging	Lee Wilcox
Modularity in Thought and Systems	Lib Gibson
The use of Domino in Optimal Resource Allocation	Ronald Frank
Interactive Statistical Graphics in APL	Neil Polhemus
Undergraduate Data Analysis Laboratory	Robert Douglas

APL as a tool of thought III – 1985	
Artificial Intelligence Development Aids	Sullivan & Fordyce
Matrix Algebra and the Mathematics of Vector Graphics	David M Laur
Relational Databases: Theory and Practice	Robert G Brown
Right Brain Thinking and Modelling Business Problems	Ron Frank
A Structured APL Approach to Data Entry	Wilbur LePage
The Design of an Integrated Manufacturing System using a Network of PCs	Clark Kee
Using APL to write an Assembler	Philip Van Cleave
APL as a Tool for Teaching Computer Theory	Philip Van Cleave
Musical Objects in Temporal Space	David Steinbrook
EXECUCALC, an APL-based Lotus 1-2-3 lookalike for IBM mainframes	Kevin Weaver
APL as a Tool concerning Accounting and Financial Simulation	Miklos Vasarhelyi
APL and Linear Algebra	L J Dickey
Teacher's Toolbox	David Michealson
Least Squares Curve Fitting Viewed Geometrically	Murray Eisenberg
Parametric and Polar Equations in Pre-Calculus Mathematics	Linda Alvord
Intensive Analysis of Global Data	Tama Traberman
Computer Simulation in Biological Education	Charles Waters
Computers and Calculus	Don Orth
Teaching Economic Concepts with APL Graphics	Robert Douglas
APL, a Tool for Teaching	Roger Pinkham
Teaching APL as a Problem Solving Tool in Business	William Royds
APL as a tool of thought IV – 1986	
APL and Application Design	Robert Bernecky
The Advatages of APL for Population Modeling	Robert Desharnais
User Friendly Applications Design	Christopher Lett
An APL Credit Card Acquisition Profitability Model	Urdang & Kaberon
Prototyping in the Real World. Considerations on a Project	Chris Oakleaf
APL in Education Engineering	LaGrega & Zaccone
Arithmetic and Geometric Progressions	Linda Alvord
APL – Alchemist's Tool	Tama Taberman
Indeterminate Error in Radiation Measurement	Charles Waters
Writing User-Friendly Applications in APL for the Apple Macintosh	Richard Smith
Using Dyalog APL	Peter Donelly
An APL to C Interface	Robert Lauer
Differences between VS APL and APL2 Release 2	Norman Brenner

The length of the articles varies considerably from a brief abstract in some cases to a full paper in others. One cannot fail to be impressed by the broad range of areas covered by the seminars and by their growing list of speakers. Further information about these proceedings, and perhaps about a 1987 seminar, can be obtained from New York SIGAPL, Suite 524, 660 Amsterdam Avenue, New York, N.Y. 10025

QL/APL User Group

Those users of QL/APL who have to date been beavering away in isolation will be interested to hear of the formation of a QL/APL User Group to exchange news, views, ideas and software.

The group will be an independent body run by users, whose aim will be to provide a vehicle for the promotion of all matters pertaining to the furtherance of APL on the QL – an environment from which to compound, augment, and proclaim hard-gained knowledge and results.

Ron Suter, who currently works for a large English mail order chain store company, has volunteered himself as founder member and consequently organiser of the initial set-up of the group, along with MicroAPL Ltd. who have offered to host the inaugural meeting.

MicroAPL have also offered a prototype copy of their MicroPlot software for the QL free of charge to anyone joining the group; like many a great symphony this is as yet unfinished, and so it offers an immediate challenge to any wishing to take up the gauntlet of group participation in the project.

In order to register your interest, please contact Ron Suter at the address given below. Please include your name and address, details of the APL version used (keyword or symbolic), amount of RAM, main areas of interest, and any suggestions for topics for User Group metings. You can contact Ron at the following address:

Ron Suter,
19 Mere Avenue,
Raby Mere, Wirral.
L63 0NE.

or leave a message on Prestel number 514285074.

IBM Personal Computer APL/PC Version 2.0

6391329

IBM Personal Computer APL/PC Version 2.0, is a low cost, full function APL interpreter with a high degree of VS APL compatibility. It contains a wealth of auxiliary processors for a wide range of functions and interfaces to external devices.

- Emulates 8087 or 80287 if the co-processor is not present
- RS232 support
- IEEE-488/GPIB support
- Co-operative processing via IBM 3278/9 adapter
- Interface to IBM Macro Assembler and Professional Fortran
- APL2 GRAPHPAK compatible workspace provided
- Can run DOS functions and applications under APL
- Cover Workspaces for auxiliary processors

The interpreter, workspaces and auxiliary processors are supplied on three double-sided diskettes packaged with a comprehensive manual, quick reference card and a keyboard template. The manual includes setup, installation, tutorial and reference sections.

A separate package is supplied, containing a replacement ROM for the IBM Monochrome or Colour display adapters and a ROM puller. A program to load the APL font into the IBM Enhanced Graphics Adaptor is also included.

- Available from Authorised IBM PC Dealers.

IBM (UK) International Products Ltd
West Cross House
2 West Cross Way
Brentford
Middlesex TW8 9DY

APL Product Guide

Compiled by Steve Lyus

VECTOR's exclusive APL Product Guide aims to provide readers with useful information about sources of APL hardware, software and services. We welcome any comments readers may have on its usefulness and any suggestions for improvements.

We do depend on the alacrity of suppliers to keep us informed about their products so that we can update the Guide for each issue of VECTOR. Any suppliers who are not included in the Guide should contact me to get their free entry – see address below.

We reserve the right to edit material supplied for reasons of space or to ensure a fair market coverage.

The listings are not restricted to UK companies and international suppliers are welcome to take advantage of these pages. Where no UK distributor has yet been appointed, the vendor should indicate whether this is imminent or whether approaches for representation by existing companies are welcomed.

For convenience to readers, the product list has been divided into the following groups:

- ★ Complete APL Systems (Hardware & Software)
- ★ APL Timesharing Services
- ★ Other services
- ★ APL Interpreters
- ★ APL Visual Display Units
- ★ APL character set printers
- ★ APL-based packages
- ★ APL Consultancy
- ★ APL Training Courses
- ★ Vendor addresses

Every effort has been made to avoid errors in these listings but no responsibility can be taken by the working group for mistakes or omissions.

Note: 'poa' indicates 'price on application'

All contributions to the APL Product Guide should be sent to:

Steve Lyus
Metapraxis Ltd.,
Hanover House
Coombe Road, Kingston
KT2 7AH

COMPLETE APL SYSTEMS

COMPANY	PRODUCT	PRICES £	DETAILS
Analogic	The APL Machine	$60,000+	AP500 array processor, 4 Mb data memory, 80 Mb disk drive.
Cocking/Drury	MicroAPL SPECTRUM	6,000	Supplied as part of a turnkey system.
	SAGE II	-35,000	See MicroAPL entry.
	SAGE IV		
Dyadic	Dyalog APL Coprocessor	3,500+	32-bit coprocessor board for IBM PC. NS32000 cpu with FPP, up to 4Mb RAM, 16Mb virtual memory. Software includes Unix V.2, Dyalog APL, graphics support, DOS interface. Provides multi-user Unix/DOS environment.
	IBM 6150	15,000+	Multi-user Dyalog APL system with Fast 32-bit RISC processor, FPP, up to 8Mb RAM, 210Mb Disk, 16 users. Interface to SQL, graphics and APL support for standard IBM peripherals.
	Altos 3068	25,000+	Multi-user Dyalog APL system with MC68020 cpu & MC68881 FPP. Also features a LAN which supports IBM PCs as Dyalog APL terminals.
	Sun 3	15,000+	Multi-user Dyalog APL systems which can be configured as a network of workstations and or a traditional time-sharing cpu. With its 25MHZ 68020 cpu, the Sun 3/200 is the fastest APL microcomputer on the market.
Gen. Software	Myriade	poa	TI computer + APL & APL operating system
Inner Product	IBM PC	2,000	IBM PCs supplied for turnkey applications
		-6,000	
M.B.T.	MBT Series 10	poa	UNIX/68010 based multi-user APL system
	TORCH	poa	68000/Z80 multiprocessor
MetaTechnics	—	poa	Details on application – IBM PC compatible
MicroAPL	Aurora	23,500	Multi-user APL computer using 68020 CPU. Std. configuration 2Mb RAM, 16 RS232 ports, 68 Mb hard disc, 720K diskette
	SPECTRUM	11,000	Expandable multi-user APL computer using Motorola 68000.
		-15,000	Std. configuration 1 Mb RAM, 12/36 Mb disc, 12 ports.
	STRIDE 440	8,500	Multi-user APL computer, 1 Mb RAM, 12/18 Mb disc.
	Atari 1040ST	799	1 Mb Mono/Colour System, includes 1 Mb disc drive & mains
		-999	transformer built into Console.

APL TIMESHARING SERVICES

COMPANY	PRODUCT	PRICES £	DETAILS
Boeing	Mainstream APL	poa	Enhanced IBM VS APL (CMS)
Mercia	APL*PLUS	poa	STSC's Mainframe Service – MAILBOX etc.
I.P. Sharp	SHARP APL	poa	International Network application systems and public databases.

APL VISUAL DISPLAY UNITS

COMPANY	PRODUCT	PRICES £	DETAILS
Dyadic	Lynwood j300	1,560	Monochrome ANSI 3.64 APL vdu, 15-inch high quality screen, Tek graphics, local macro keys.
	Lynwood j500	2,295	Colour ANSI 3.64 APL vdu, 15-inch high quality screen, Tek graphics, local macro keys.
	IBM 3163	791	Low-cost Monochrome APL vdu. Supports downloaded Dyalog APL font.
	IBM 3164	1,093	Low-cost Colour APL vdu. Supports downloaded Dyalog APL font.
Farnell	Tandberg TDV 2221	995	Ergonomic design APL terminal, 50-19200 baud, 15" anti-reflex screen, low profile keyboard
	Tandberg TDV 2271	1,195	Combined APL/ANSI ergonomic terminal as above.
Gen. Software	Mellordata Elite 3045A	400	Second-hand
M.B.T.	various		Contact MBT for details
Meta Technics	IBM EGA compatible	299	Emulates EGA & Hercules, Half Card
MicroAPL	Insight VDT-1	795	Inexpensive APL VDU
	Insight GDT-1	1,450	With monochrome graphics
	Concept 201	1,295	APL VDU with 8 page memory
	Concept 201G	1,650	Graphics VDU
Shandell	HDS2010	1,215	ANSI X3.64 compatible, full overstrike chars., 4/8 pages, 2/3 comms. ports, 80/132 cols., windowing, viewports, 15" screen.
	HDS2010G	1495	As HDS2010 plus Tektronix 4010/4014 graphics with 1024 x 390 resolution.
	HDS2010GX	1,775	As 2010G but with 1024 x 780 resolution.
	HDS2210	1,215	DEC VT220 compatible, full overstrike characters, 4 page memory, 2/3 comms. prts, 80/132 cols., 15" screen
	HDS2210G	1,495	As HDS2210 plus Tektronix 4010/4014 compatible graphics with 1024 x 790 resolution. Also additional capabilities of Retrographics VT640/DQ640 and Visual 500 terminals.
	HDS2210GX	1,775	As HDS2210G but with 1024 x 780 resolution.
Textronix	4114B	13,500+	19" D.V.S.T.:Graphics: 3120 x 4096 displayable; Intelligent: up to 800K memory; APL keyboard (option 4E)
	4125	21,550+	19" 2D colour graphics; Workstation (1280 x 1024);Intelligent: up to 800K memory; APL keyboard (mod AP)
	4128	26,822+	As 4125 plus 3D wireframe

APL PRINTERS

COMPANY	PRODUCT	PRICES £	DETAILS
Datatrade	Datasouth DS180+	1,295	180 cps matrix printer with 4K buffer, 9 x 7 dot matrix and APL option.
	Datasouth DS220	1,695	Letter quality; graphics capability, APL option (both available with IBM Twinex or Coax interface).
Dyadic	IBM 4201 Proprinter	poa	100, 200, 40(nlq) cps, matrix printer, with graphics. Supports downloaded Dyalog APL font.
	Toshiba P351	poa	24 pin high-quality matrix printer 100 cps letter quality, 192 cps draft.
Inner Product	Epson FX80	500	Soft char. set, 160 cps, 80 column
	Anadex 9620	1,150	200 cps., 132 col., tractor feed
	Siemens PT88	620	180 cps., 80 col., silent
	TGC Starwriter	1,180	40 cps., letter quality
M.B.T.	Facit 4565	poa	40 cps letter-quality
	Facit 4510/11/12	poa	Matrix printers
MetaTechnics	Quen-data	295	Low-cost APL Daisy-wheel printer
MicroAPL	Datasouth DS180+	1,295	See Datatrade entry
	Philips GP300	1,924	Matrix printer with letter & draft quality and APL.
	Qume Letterpro20	549	APL/ASCII Daisy-wheel printer

OTHER PRODUCTS

COMPANY	PRODUCT	PRICES £	DETAILS
APL People	Employment Agency	poa	Permanent employees placed at all levels. Contractors supplied for short/long-term projects, supervised.
Mine of Information	APL Book Service		See booklist
I.P. Sharp	Productivity Tools	poa	Utilities for systems, operations, administration & analysts; auxiliary processors, comms software, international network.
	Databases	poa	Financial, aviation, energy and socioeconomic.

APL PACKAGES

COMPANY	PRODUCT	PRICES £	DETAILS
APL◇385	FSM 385 PC:	50	Screen development
	DRAW 385		Screen design
	DB 385 Mainframe:	125	Relational W.S.
	GEN 385		Utilities
APL Software Ltd	*Mainframe*		
	AFM/AP	11,035	Interprocess Software for VM/CMS & MVS/TSO.
	– Keyed Access	2,650	Component File Management System (VSAPL/APL2)
	– Interactive Link	1,325	
	– Mail Exchange	2,650	
	CALL/AP	4,030	Non-APL program execution (VSAPL/APL2)
	APLPRINT	2,205	Output to high speed line printer or 328x devices (VSAPL/APL2)
	ENHANCED FORMAT	2,205	Extends Format operator to full "Quad-FMT" status (VSAPL/APL2)
	ISP	750	Input and Output Stack Processors for manipulating terminal I/O
	OSP	2,205	with facilities for Error Trapping (VSAPL)
	DISPLAY CAPTURE	poa	Allows terminal output to be collected and held for retrieval by an APL function (APL2)
	UCF	poa	User Communication Facility for data transfer between users (APL2)
	RDS	poa	Relation Data Base System
	PANEL	poa	Fullscreen management system
	PFS	poa	Program File System – APL Systems development aid
	Microcomputer		
	POWERTOOLS	295	Assembler written replacement function for commonly used CPU-consuming APL functions, includes a Forms Processor.
Beta-plan	BETA-FONT	poa	Multiple font PC character generator. Dealers required for non-Scandinavian countries.
Boeing	TABAPL	poa	Hierarchical Planning System
Butel	Merlin	5,000	Mainframe APL spreadsheet runs under VM/CMS, TSO, VSPC
	Merlin/PC	poa	Version for APL*PLUS/PC
Cocking/Drury	*Mainframe*		
	STSC's SHAREFILE & enhancements to VSAPL	poa	Component files, quad-functions & nested arrays for IBM VSAPL under VM/CMS & MVS/TSO
	SHAREFILE AP	15,000	STSC's sharefile for APL2
	FILEMANAGER	poa	STSC's database package.
	FORMAT	2,250	Enhanced report formatting.
	COMPILER	30,000	First APL compiler. Available with APL*PLUS enhancements and Sharefile under VM/CMS & MVS/TSO
	SQL	poa	An interface between SQL and APL*PLUS for VSAPL
	FMT	poa	Full featured FMT for APL2
	DATAPORT	poa	Powerful Information Centre spreadsheet incorporating data exchange between APL and FOCUS, IFPS, SAS, APL/DI, ADRSII, LOTUS123, VISICALC, MULTIPLAN, DIF files
	Microcomputer		
	STATGRAPHICS Rel 2	545	Powerful Statistics and graphics on IBM PC's, PC/AT's and compatibles
	Release 2 update	165	Update from release 1 to release 2
	APL*PLUS PC Tools		

		VOL 1	325	Incl. 327x IRMA support, RAM disk, full screen data entry, menu input, report generation, games.
		VOL 2	125	Incl.file documentor, screen editor, exception handler.
	APL*PLUS PC Fin & Stat. Library		350	Financial & statistical routines
	SPREADSHEET MANAGER		195	APL-based spreadsheet for APL*PLUS/PC. Cell arithmetic; transfers to ASCII, LOTUS
E&S	PROTOPAK consisting of:			Packages for prototyping management information systems – PC & mainframe
	RMS	Modules		Relational databases.
	AMS		250+	Multi-dimensional arrays
	RAMS			Combined RMS & AMS.
	BMS			Dynamic financial modelling & forecasting
	FMS			Full-screen handler for APL*PLUS/PC. (AP 124-based)
	CMS			Communications package.
	SOS		poa	Scheduled ordering and stock control.
Gen. Software	PROPS		500+	Spreadsheet system for Product and/or Project Planning.
H.M.W.	INPUT		poa	Matrix manipulation package for data entry & report generation
	PRINTPAK		poa	Block printing for VM/CMS
	VIEWPAK		poa	AP124 Protocol emulator for IBM/PC
Holtech	CASH		3,500 –10,000	Accounting package & hotel management system on MicroAPL SPECTRUM & SAGE CPUs.
Inner Product	Viewcom		150	Control Viewdata from APL
	APL/dBASE II		150	Interface APL with dBase II
	APL/dBASE III		150	Interface APL with dBASE III
	APL/LOTUS		150	Interface APL with Lotus
	APL/WORDSTAR		150	Interface APL with Wordstar
	APL/MULTIPLAN		150	Interface APL with spreadsheet
	CEMAS		3,500	EEC monetary and agrimonetary analysis.
M.B.T.	RHOMBUS		poa	Integrated Office System
	HASLEMERE		poa	Hotel Accounting System
Mercia	STATGRAPHICS 2		535	Integrated stat. graphic system for PCs.
	Upgrade to Release 2		175	
	EXEC*U*STAT		425	Easy to use Statistics for management.
	APL*PLUS tools			
		VOL 1	225	IBM PC Utilities:IRMA3270 comms, full screen, RAM Disk report generator
		VOL 2	125	File documentation, screen editing. Exception handling.
	FINANCIAL AND STATISTICAL LIB.		325	Financial and Statistical analysis
	INFO CENTRE		2,000 –20,000	Full-screen entry, display & multi-dimensional analysis. Interfaces to other I.C. products. Runs under VM VSAPL on IBM mainframes.
	APL Spreadsheet Manager		195	APL spreadsheet – links to popular spreadsheet software.
	APL Debugger		195	Powerful debugging tool for APL*PLUS/PC
	UNITAB		495	Spreadsheet for APL*PLUS/PC
	MULTI-APL		poa	Multi-user/Multi-tasking APL*PLUS/PC
	EXECUCALC		4,000	Mainframe Spreadsheet with VisiCalc and Lotus 1-2-3 functionality requires VSAPL under TSO or VM.
	EXECUPLOT		3,200	Mainframe Graphics display system with VisiPlot functionality requires VSAPL under TSO or VM and GDDM.
	MICROSPAN		250	Comprehensive APL tutor

MetaTechnics	MetaScreen	99	Full-screen handler for APL*PLUS/PC, based on VSAPL AP124
	MetaPack	495	Comprehensive utilities package for APL*PLUS/PC. Includes MetaScreen, MetaWS, Browse, Toolbox, Numeric Editor.
	APL-IEEE488	99	Controls IEEE488/GPIB Bus from APL*PLUS/PC.
	PLOT/PC	99	2D & 3D Graphics package. Includes interactive diagram Editors.
	Browse	99	Scrolling of DOS files, large APL variables.
	ADAPTA DLS	poa	Production & purchasing scheduling for process manufacturing.
	ADAPTA MSP	poa	Job-shop loading & scheduling for multi-stage production.
MicroAPL	MicroTASK	250	Product development aids
	MicroFILE	250	File utilities and database
	MicroPLOT	250	Graphics for HP plotters etc
	MicroLINK	250	General device communications
	MicroEDIT	250	Full screen APL editor
	MicroFORM	250	Full screen forms design
	MicroSPAN	250	Comprehensive APL tutor
	MicroGRID	poa	Ethernet & other networking
	APLCALC	400	APL spreadsheet system
	MicroPLOT/PC	250	For APL*PLUS/PC product
	MicroSPAN/PC	250	For APL*PLUS/PC product
	PC TOOLS Vol 1	295	
	STATGRAPHICS Rel 1	495	
	STATGRAPHICS Rel 2	535	
Parallax	ExecuCalc	$5,000	Mainframe-based electronic spreadsheet for VM/CMS & MVS/TSO with links to micro products.
	ExecuPlot	$5,000	Mainframe-based colour graphics with micro links.
I.P. Sharp	ACT	poa	Actuarial system
	APS	poa	Financial Modelling
	BOXJENKINS	poa	Forecasting technique
	CONSOL	poa	Financial Consolidation
	COURSE	poa	APL Instruction
	EASY	poa	Econometric Modelling
	FASTNET	poa	Project Management
	GLOBAL LIMITS	poa	Exposure management for banks
	MABRA	poa	Record maintenance/reporting
	MAGIC	poa	Time series analysis/reporting
	MAGICSTORE	poa	N-dimensional database system
	MAILBOX	poa	Electronic Mail
	MICROCOM	poa	Mainframe to micro link
	SAGA	poa	General graphics, most devices
	SIFT	poa	Forecasting system
	SNAP	poa	Project management
	SUPERPLOT	poa	Business graphics
	VIEWPOINT	poa	4GL – Info centre product
	XTABS	poa	Survey Analysis
Sugar Mill	Stat 1	$129.95	Statistical toolbox, menu driven

APL CONSULTANCY

(prices quoted are per day unless otherwise marked)

COMPANY	PRODUCT	PRICES £	DETAILS
APL People	Consultancy	poa	Project management, financial applications, relational databases. Difficult problems solved. Management consultancy. Links to non-APL systems. From consultant level to managing consultant. Documentation a speciality.

Boeing	Consultancy	poa	
Camacho	Consultancy	poa	Specialising in programming & manual writing.
Cocking/Drury	Consultancy	120-150 140-200 185-300 275-400	Junior consultant Consultant Senior consultant Managing consultant
Delphi	Consultancy	poa	Specialising in management reporting systems and APL on microcomputers.
Dyadic	Consultancy	poa	APL system design, consultancy, programming & training for Dyalog APL, VSAPL, APL*PLUS, IPSA APL etc.
E&S	Consultancy	150 -250	System prototyping: all types of information system.
FASTCODE	Consultancy	poa	Specialise in improving performance of APL applications on micros & mainframes.
Gen. Software	Consultancy	100+	
H.M.W.	Consultancy	100-250	System design consultancy, programming.
Inner Product	Consultancy	200	On-site micro-mainframe APL, PC/DOS & Assembler
Lloyd Savage	Consultancy	poa	Decision support, particularly specialising in Sales & Marketing systems.
M.B.T.	Consultancy	poa	
Mercia	Consultancy	poa	APL*PLUS & VSAPL consultancy.
MetaTechnics	Consultancy	poa	Management Information & Production. Engineering APL – C/Assembler custom programming
MicroAPL	Consultancy	poa	Technical & applications consultancy.
M.T.I.	Consultancy	poa	Specialise in Maintenance and development of existing APL systems
Parallax	Consultancy	$750	Introductory APL, APL for End-user & Advanced Topics in APL
QB On-Line	Consultancy	200	Specialising in Banking, Financial & Planning Systems.
Rochester Group	Consultancy	poa	Specialise in MIS using Sharp APL
I.P. Sharp	Consultancy	poa	Consultancy & support service world-wide.

APL INTERPRETERS

COMPANY	PRODUCT	PRICES £	DETAILS
Cocking/Drury	APL*PLUS/PC Rel 6	475	STSC's full featured APL for IBM PC, PC/AT and compatibles
	Upgrade 5 to 6	120	Extension from rel 5 which incorporates 64K object support.
	Upgrade 2,3,4 to 6	225	Extension upgrades to release 6.
	Run-time	poa	Closed version of APL*PLUS/PC which prevents user exposure to APL.
	APL*PLUS UNIX	poa	STSC's 2nd generation APL for IBM PC/AT, DEC, AT&T and other Unix computers.
Dyadic	Dyalog APL	795 -10,000	2nd gen. APL for UNIX systems, e.g. IBM 6150, Sun, Vax, NCR, HP9000, AT&T, Altos, Apollo, Whitechapel, Sperry, etc.
Gen. Software	APL*MYRIADE	poa	Runs on Texas Instruments TI990 range.
IBM UK Product Sales	IBM PC APL	poa	Event-handling & APs for full-screen I/O disks, diskettes, asynch. comms.
Inner Product	VIZ::APL	250 -350	8-bit Zilog Z-80 CP/M
	APL*PLUS/PC	600	See under Cocking & Drury

M.B.T.	Dyalog APL	poa	See Dyadic Systems entry
	MBTAPL	poa	Enhanced Dyalog APL for MBT hardware.
	VIZ::APL	poa	Customized for TORCH hardware
Mercia	APL*PLUS/PC Rel 6	495	STSC's full-feature APL for IBM PC, PC/AT Compaq, Olivetti, Wang, Apricot, Ericsson etc
	Upgrades 3 to 4	100	
	Upgrades 4 to 5	150	
	Upgrades 5 to 6	130	
	APL*PLUS/UNIX	poa	Interpreter for UNIX systems: WICAT, CADMUS, CALLAN, FORTUNE 32:16, HP, 9000/500, OLIVETTI 3B2
MetaTechnics	APL*PLUS Rel 6	475	Discount on quantity.
MicroAPL	APL.68000	1,000+	Full implementation with component files, error trapping etc. for SPECTRUM, SAGE & other MC68000-based computers.
	QL/APL (keyword)	87	Full keyword APL for QL with many extra features.
	QL/APL (APL chars)	87	VSAPL compatible APL for QL with many extra features.
	APL.68000 for Apple Macintosh	257	
	APL.68000 for Commodore Amiga	200	
	APL.68000 for Atari ST	170	
	APL*PLUS/PC – REL 6	450	
Portable	PortAPL	$195	IBM PC Software
		$275	Mackintosh
		$2,995	DEC VAX
I.P. Sharp	Sharp APL/PCX	2,575	For IBM XT/AT
		1,000+	For IBM mainframes
	Sharp APL/PC	325	For IBM PC or PC/XT

APL TRAINING COURSES

(Prices quoted are per course unless otherwise stated)

COMPANY	PRODUCT	PRICES £
Cocking/Drury	3 day Fundamentals	375
	4 day APL*PLUS/PC Intermediate	525
	5 day System Design	595
	Introduction to APL2	poa
	APL2 in Depth	poa
Inner Product		poa
M.B.T.		poa
Mercia	3 day Introduction to APL	350
	2 day APL*PLUS/PC Enhancements	240
	3 day APL*PLUS System Design	375
Parallax		poa

VENDOR ADDRESSES

COMPANY	CONTACT	ADDRESS & TELEPHONE No.
Analogic Corporation	Denise Favorat	8 Centennial Drive, Centennial Industrial Park, Peabody, Mass. U.S.A. 01961 ☎ 617-246-0300
APL 385	Adrian Smith	Brook House, Gilling East, York. ☎ 04393-385
APL People	Valerie Lusmore	17 Barton Street, Bath, Avon. ☎ 0225-62602
APL Software Ltd	Philip Goacher	27 Downs Way, Epsom, Surrey KT18 5LU ☎ 03727-21282 17 Barton Street, Bath, Avon BA1 1HQ ☎ 0225-62602
Beta-plan APS	Kim Andreasen	Stengrade 75 , DK-3000 Helsingor, Denmark. ☎ 45 2 21 48 48
Boeing Computer	Anne Harding	19 Fitzroy Street, Services (Europe) Ltd., London W1P 5AB. ☎ 01-631 0808
Butel Technology Ltd.	Mike Munro	Butel House, 3 Great West Rd., London W4 5QJ ☎ 01-995-1433
Anthony Camacho		2 Blenheim Road, St. Albans, Herts AL1 4NR. ☎ St. Albans 60130
Cocking & Drury Ltd.	Romilly Cocking Brian Drury	16 Berkeley Street, London W1X 5AE. ☎ 01-493 6172 155 Friar Street Reading RG1 1HE. ☎ 0734-588835
Datatrade Ltd.	Tony Checksfield	38 Billing Road, Northampton, NN1 5DQ. ☎ 0604-22289
Delphi Consultation Ltd.	David Crossley	Church Green House, Stanford-in-the-Vale, Oxon SN7 8LQ. ☎ 03677-384
Dyadic Systems Ltd.	Peter Donnelly	Park House, The High Street, Alton, Hampshire. ☎ 0420-87024
E & S Associates	Frank Evans	19 Homesdale Road, Orpington, Kent BR5 1JS. ☎ 0689-24741
Farnell International Instruments Ltd.	R. Fairbairn or Roger Attard	Jubilee House, Sandbeck Way, Wetherby, W. Yorks. ☎ 0937-61961 Davenport House, Bowers Way, Harpenden, Herts. ☎ 05827-69071
FASTCODE	Andrew Dickey	P.O. Box 281, Croton-on-Hudson, New York 10520, U.S.A. ☎ (914) 271-3200
General Software Ltd.	M.E. Martin	22 Russell Road, Northolt, Middx. UB5 4QS. ☎ 01-864 9537
H.M.W. Programming Consultants Ltd.	Ken Jackson	142 Feltham Hill Rd, Ashford, Middx. TW15 1HN. ☎ 07842-41232
Holtech Ltd.	Jan Bateman	'O' Block 4th Floor, Metropolitan Wharf, Wapping Wall, London E1 9SS. ☎ 01-481 3207
IBM UK Ltd	Chris Sell	PO Box 32, Alencon Link, Basingstoke, Hants. RG21 1EJ. ☎ 0256-56144
Inner Product Ltd.	Dominic Murphy	Eagle House, 73 Clapham Common Southside, London SW4 9DG. ☎ 01-673 3354
Lloyd Savage Ltd	Philip Johnson	Cambridge House, Oxford Road, Uxbridge. Middx, UB8 2UD. ☎ 0895-59826
Mercia Software Ltd.	Gareth Brentnall Barrie Webster	Aston Science Park, Love Lane, Birmingham B7 4BJ. ☎ 021-359 5096
MetaTechnics Systems Ltd	John Stenbridge David Toop	Unit 216, 62 Tritton Road, London, SE21 8DE. ☎ 01-670 7959
MicroAPL Ltd.	Bernadette Leverton	19 Catherine Place, London SW1E 6DX ☎ 01-834 9022
Mine of Information	Richard Ross-Langley	PO Box 1000, St. Albans, Herts AL3 6NE. ☎ 0727 52801
Modern Business Technology Ltd. (MBT)	Michael Branson	P.O. Box 87, Guildford, Surrey GU4 8BB ☎ 04868-23956
M.T.I.	Ray Cannon	7 Pine Wood, Sunbury-on-Thames, Middx. TW16 6SH ☎ 09327 80848
Parallax Systems Inc.	Kevin Weaver	60 West 9th Street, New York, New York 10011, U.S.A. ☎ 212-475-4001
Portable Software	Richard Smith	60 Aberdeen Ave, Cambridge, Mass. U.S.A. 02138. ☎ 617-547-2918
QB On-Line Systems	Philip Bulmer	5 Surrey House, Portsmouth Rd Camberley, Surrey, GU15 1LB. ☎ 0276-20789
The Rochester Group	Robert Pullman	164 Pinnacle Rd., Rochester NY 14620 ☎ 716-461-3169
Shandell Systems Ltd.	Maurice Shanahan	12 High Street, Chalfont St. Giles, Bucks HP8 4QA. ☎ 02407-2027
I.P. Sharp Associates Ltd.	David Weatherby	10 Dean Farrar Street, London SW1. ☎ 01-222 7033
Sugar Mill Software Corp.	Lawrence H. Nitz	1180 Kika Place, Kailua, Hawaii 96734 ☎ (808) 261-7536
Tektronix UK Ltd.	Paul Morgan	Fourth Avenue, Globe Park, Marlow, Bucks SL7 1YD. ☎ 06284-6000

APL BOOKLIST
(In author order)

	Title	Price £	UK P&P
*	Sharp APL Reference Manual, P Berry	10.50	2.70
	Star Map, P Berry & J Thorstetsen	6.00	.50
	APL and Insight, P Berry and G Bartoli	4.50	.55
	APL 86 Tutorials, A Camacho (members £9.30)	12.00	2.00
*	A Source Book in APL, A Falkoff & K Iverson	10.00	1.80
*	FinnAPL Idiom Library (when available)	11.20	1.30
	Application Systems in APL, Gibson Levine Metzger	30.00	2.70
*	APL:An Interactive Approach, Gilman & Rose	26.50	2.75
	Solutions to Algebra, J Iverson	3.00	.50
	A Dictionary of APL, K Iverson	2.50	.50
	A Concise Dictionary of APL, K Iverson	2.00	.50
	Algebra: an Algorithmic Treatment, K Iverson	22.50	2.30
	APL in Exposition, K Iverson	3.00	.50
	Applied Mathematics for Programmers, K Iverson	8.00	1.35
	Elementary Analysis, K Iverson	8.00	1.50
	Introduction to APL for Scientists & Engineers, K Iverson	3.00	.40
	Introducing APL to Teachers, K Iverson	3.00	.40
	Mathematics and Programming, K Iverson	8.00	.95
	APL Toolkit (CIPS APL SIG), R Levine	4.50	.95
	Reliable Software Through Composite Design, Myers	15.00	1.30
	APL:An Introduction, H Peelle	POA	
*	APL The Language & its Usage, R Polivka & S Pakin	36.35	2.70
	APL in Practice, Rose/STSC	40.00	2.50
	Sharp APL Users Meeting Procs 1984 (Information Centres)	8.00	1.80
	Sharp APL Users Meeting Procs 1982 Vol 2	8.50	1.80
	Sharp Pocket APL Reference Book	1.50	.50
*	APL:Design Handbook for Commercial Systems, A. Smith	13.10	1.50
	Resistive Circuit Theory, R Spence	20.00	2.80
	Whizzbangs Volume II, R Sykes	14.50	1.80
	Whizzbangs Volume I, R Sykes	14.50	.40
	An APL Notebook, Barrie Wetherill (when available)	1.90	.50
	APL Idiom List (Yale University)	2.00	.50
	APL 86 Conference Procs, D Ziemann (members £13.30)	17.30	2.25
	APL Business Technology '83 Proceedings	11.20	2.70
	APL Lapel Pin with gripping backplate	2.00	.50
	APL Quote-quad the Early Years	32.00	2.70
	APL SV Reference Card (Vade mecum)	.50	.50
	APL Trivia Cards (per set)	4.50	.60
	Special APL 86 Wallet Offer (see Vector 3.2 page 92)	22.50	5.00

PLEASE ORDER DIRECT FROM MINE OF INFORMATION

APL Book Service, PO Box 1000, St Albans, AL3 6NE, UK
Telephone 0727-52801
Prices are subject to change without notice
Access and Visa accepted, or Sterling cheque with order.

Outside UK add an EXTRA amount per book-
£1 to Europe, £2.50 Africa and Middle East, £5 elsewhere (all sent airmail)

* In a poll of the British APL Association Committee the books marked with an asterisk were highly thought of among those they had read. Vector hopes to publish a set of potted book reviews for all books in stock soon.

The Education VECTOR

by Norman Thomson

The previous column under this heading concluded by announcing how the idea of Public Domain APL was born at APL86. Since then, I-APL (International APL) has been born, and the embryonic concept has moved inexorably to project status. The British APL Association has promised funds subject to committee approval by 1st December of a technical specification and business plan. SIGAPL has promised similar support. An appeal has been made to individual and sustaining members of the BAA and to the other European associations to contribute to the £30,000 which is required to see the project to successful completion. Happily enough initial funds have been guaranteed to enable the project to engage Paul Chapman to embark on the I-APL interpreter, while on both sides of the Atlantic work is progressing with the supporting documentation.

The existence of an interpreter and documentation is of course by no means all. The next stage will then be to bombard our schools with APL, and although our plans include approaches to LEA Mathematics Advisers, Mathematics Project Directors and High IQ (Information Technology Quotient) Schools, this is a point at which **all** members of the Association have a part to play in consciously telling and persuading the rest of the world that APL is here, and available to all, and that it contains the seeds of important educational innovation in the Mathematics and Science classroom. In particular those members who are concerned with recruitment into their companies have a special role to play. An argument heard regularly from Polytechnic lecturers is that they do not teach APL because it is not used by employers. How I wish I could broadcast the telephone conversation I had recently with a representative of a large British company asking me if I knew at which Polytechnics they should look to recruit graduates competent in APL. Is there a turning point in view?

Meanwhile we continue to hear of new APL enterprise in education. At University College, London, for example, Dieter Girmes in the Statistical Science Department runs a 2nd/3rd year course on Mathematical Computation which is based entirely on APL. After a minimum of formal instruction, students are issued with sheets each containing an APL function and its associated "brother" function which gives a detailed trace of execution, thereby explaining what the various APL primitives are doing. The aim is not to impart any new mathematics, but rather to relate known mathematics and statistics to APL, and thereby focus on the power of APL to express and execute algorithms. The course is practical rather than didactic, and support is provided by terminals connected to the College's Euclid computer.

On the school front, delegates at the MUSE (Microcomputer Users in Schools and Education) Conference showed considerable interest in APL, and we hope that by the time of the next conference I-APL will be a viable and attractive option on school micros. At Portsmouth Grammar School, the computer donated by Quaker Oats in the competition judged by the BAA has been received with enthusiasm by pupils, one of whom has been working on a mini-APL interpreter in COMAL as a project.

Meanwhile, as the dawn of I-APL is awaited, we take a look at what low-cost APL systems are currently available to the home or school user. First we mention two free APLs – and yes we do mean free! These are VIZ::APL – put into the Public Domain by Inner Product and which runs on the RML 380Z and other CP/M machines – and Acornsoft APL which runs on the BBC micros equipped with the second Z80 processor. Anthony Camacho can be contacted for details of the Acornsoft interpreter and we hope to have VIZ::APL available through the BAA by the end of the year; please do not ask Inner Product for copies.

The cheapest APL system remains the Sinclair QL, which can still be obtained; ask MicroAPL if you have problems. This typically costs around £200 with a monitor, and QL/APL from MicroAPL costs a further £100 in either keyword or symbolic notation. (These prices in common with all the others quoted here include VAT.) QL/APL is a full-blown no-corners-cut APL with file facilities and graphics links. I have been using it myself at home for a year or so and I have yet to find any flaws in the interpreter.

The next available product going up the quality ladder is the Atari 1040 for which you can expect to pay £920 with a monochrome monitor, or £1150 with colour. APL (again from MicroAPL) will cost just under £200. APL for the Commodore Amiga comes in at £260, and for the Apple Macintosh at £295. MicroAPL reports shipments of 300 per month for this product, but sadly most of these are to the Continent. PortaAPL, the other APL product which runs on the Macintosh, sells for about $200. (Ed: most of these interpreters have been reviewed in recent issues of VECTOR.)

If your budget runs to the IBM PC or its look-alikes, the best purchase is version 2 of IBM's own APL which retails at £328 and is subject to its Educational discount. STSC's longer-established APL*PLUS sells at £546-25; Pocket APL is STSC's mini-version of APL, which costs about $100 but is not on sale in the U.K. It is too early to say whether the new Amstrads will run APL successfully, but the first signs look hopeful.

In summary, APL is poised at an educational cross-roads – I-APL promises the opportunity to prove once and for all that it is only cost which has prevented APL from taking its place as one of the most popular and useful computer languages. We move forward in faith and hope.

REVIEWS SECTION

For the first time, this issue of VECTOR contains a separate section devoted to reviews. This section will contain reviews of books, software and hardware of interest to the APL community at large.

Val Lusmore of APL People has agreed to take on the role of Reviews Editor to coordinate the reviewing process. Whilst the VECTOR team try to cover as wide an area as possible, we cannot reasonably expect to notice every new product. Val would therefore welcome input from three sources:

- product vendors who are able to make available product for review;
- ideas for books and products suitable for review;
- volunteers who would like a chance to evaluate new products and write a review for VECTOR.

All ideas should be sent to:

Val Lusmore,
APL People Ltd.,
17 Barton Street,
Bath, Avon
Tel. 0225-62602

This issue covers three book reviews and two APL implementations. The first book review is a double-header; Simon Garland reviews two recent books by Ken Iverson – "Applied Mathematics for Programmers" and "Mathematics and Programming". Another renowned APL author is Howard Peelle, and his "Introduction to APL" is reviewed by Romilly Cocking. Thirdly Peter Branson gives us his thoughts on reading "A Handbook and Guide for Comparing Computer Languages", produced by the Research and Education Association of New York.

Paul Chapman is an independent consultant, who is currently working for the I-APL committee writing a free public-domain APL interpreter for small computers. In his copious free time Paul kindly agreed to review MicroAPL's APL.68000 implementation on the Atari 520 ST.

Finally Martyn Adams of Metapraxis offers some first impressions on the latest release (6.0) of STSC's APL*PLUS/PC, including a fleeting and evidently unsatisfactory encounter with the new Compaq 386 microcomputer.

Book reviews

Applied Mathematics for Programmers (157 p)

Mathematics and Programming (132 p)

by Kenneth E. Iverson

Published by I P Sharp Associates, 1986.

Reviewed by Simon Garland

Both of these new self study courses from Dr. Iverson use an executable notation, that is (as he explains in the introduction to *Mathematics and Programming*) . . a notation whose rules are so strict and simple that a reader, or computer, can interpret any expression unambiguously.

The executable notation used is APL, as defined in *A Concise Dictionary of APL* (also available from I.P.Sharp). No prior knowledge of the notation is assumed, it is introduced clearly and simply as required – in refreshing contrast to the usual half apologetic jokes about strange symbols and unreadable one-liners.

In order to be able to use the course texts easily the reader should have access to a union keyboard version of APL and a copy of a direct definition workspace that uses the

FunctionName : ElseExpression : IfCondition : ThenExpression

form of definition.

As far as I could tell all examples could be executed without problem (apart from the response time..) with the current version of Sharp APL/PC. APL*PLUS/PC users will not be able to use any of the expressions using Sharp enhancements like the rank operator, or enclosed arrays, but they can still execute most of the expressions without much problem by simply replacing the lev and dex symbols by the functions:

```
    ∇ Result←Result LEV Junk
[1]   ⍎ ⍵
    ∇

    ∇ Result←Junk DEX Result
[1]   ⍎ ⍵
    ∇
```

Better still make the direct definition compiler do it (exercise -1).

The aim of *Applied Mathematics for Programmers* is to make the mathematics involved in many of the basic tools of programming more accessible to programmers – to enable them to grasp the concepts behind the tools they use.

Anyone who has had to sit through a lecture on how quicksort works, complete with helpful diagrams, examples, handwaving, and the exhortation that it's all very simple will appreciate the power of a concise executable notation that allows the student to experiment with the simple functions:

```
Sort: (B/⍵),Sort(~B←⍵=⌊/⍵)/⍵ : 0=⍴⍵ : ''

QS: (QS(⍵<F)/⍵) , ((⍵=F)/⍵) , (QS(⍵>F)/⍵)←F←1↑⍵ : 0=⍴⍵ : ''
```

Mathematics and Programming is more a review of high school mathematics using an executable notation, which exposes and clarifies relations among topics previously studied in isolation; it also goes into more detail about the notation and its 'grammar'.

Both courses go over similar ground, but with different emphasis. They cover classifications and sets, the behaviour of elementary functions (utility functions are supplied for drawing graphs and barcharts to assist with the exploration of the behaviour of the functions), directed graphs and trees, identities and proofs, and modelling (including writing your own computer). Exercises to check and extend undertanding of the points under discussion are provided at the end of the book, with indicators within the text when an exercise is available.

Applied Mathematics for Programmers has a special section on manual translation with examples of translation to Pascal, C, Fortran, and Conventional Mathematical Notation, closing with a discussion on general translation techniques. This chapter is an example of the attitude that permeates the whole course – no messianic preaching APL to the masses, just a demonstration of how useful it is to be able to translate ideas into a concise clear notation to help with the understanding and communication of concepts – if a program works in Fortran then fine, don't touch it, but perhaps it would be easier to document the essential algorithm in something less cumbersome.

Mathematics and Programming has a special section on co-ordinate geometry – I didn't find this as interesting, but I've probably been scarred for life by not having courses like this one. . .

Although these books are meant as course texts I think most APL users could profit from a careful study of the examples. Consider for example one of the identities discussed under Efficiency in the chapter on Identities and Proofs in *Applied Mathematics for Programmers*; the problem is the common one of applying an expensive function F to a vector of non-distinct values v. The brute force solution (any mistakes in the normal APL examples are my fault) is something like:

```
...
[10]       r←(rv←⍴v)⍴0
[11]      →lc←(rv⍴1020)),1022→i←1
[12]1020: r[i]←F v[i]
[13]1021:→lc[i←i+1]
...
```

Of course we could 'optimise' this considerably to:

```
...
[42]       r←(rm←1⌷⍴msk←(uv←Nub v)∘.=v)⍴0
[43]       cols←⍳0⌷⍴msk
[44]      →lc←(rm⍴1030),1032→i←1
[45]1030: r[msk[i;]/cols]←F uv[i]
[46]1031:→lc[i←i+1]

    (Nub: ((⍳⍴⍵)=⍵⍳⍵)/⍵)
...
```

but with a little more thought we could have written it much more clearly from the beginning (if it's not clear then try it out!):

```
...
[1] r←(n∘.=v)+.×F n←Nub v
    ∇
```

Most of all I would hope that those responsible for teaching programming (not just those responsible for teaching APL..) or mathematics would take the time to look carefully at these texts with a terminal nearby – I'm certain that students would approach such courses with much more enthusiasm than a standard lecture oriented course.

Finally a quote from C.A.Hoare, reported in the August 1986 edition of BYTE magazine:

> "Many programmers of the present day have been educated in ignorance and fear of mathematics. Of course, many programmers are mathematical graduates who have acquired a good grasp of topology, calculus and group theory. But it never seems to occur to them to take advantage of their mathematical skills to define a programming problem and search for its solution.
>
> "Our present failure to recognize and use mathematics as the basis for a programming discipline has a number of notorious consequences. They are the same as you would get from a similar neglect of mathematics in drawing maps, marine navigation, bridge building, air-traffic control, and exploring space."

An Introduction to APL

by Howard A Peelle

Published by Holt, Rinehart and Winston, 1986.

Reviewed by Romilly Cocking

In the early Seventies, there were not many ways of learning APL. This did not matter much, as very few people wanted to learn it. Most students of the language were unwilling. They were victims of zealots who had just become APLers, and who were determined to share the good news.

Today, the situation is much healthier. A growing number of companies are making a healthy profit from selling APL products and services; more important, a large and growing number of organisations now depend on the use of APL to help them achieve their goals.

APL is in demand, and more people want to learn it than ever before. A wide variety of methods are available – public courses, CAI programs, and self-study books. For some years, the standard text for introductory courses has been Gilman and Rose (3rd edition, Wiley 1984). Originally published in 1974, 'the red book' has gone through three editions, changed its colour, and sold well over a quarter of a million copies.

No book is ideal, and APL trainers are always on the lookout for new texts. I have just been reading *APL – an introduction*, by Howard Peelle. The text is intended for two uses; it can be read as a self-study guide, for individual students who are learning APL on their own; alternatively, it can be used as part of a taught course. The text has an associated volume of instructor notes, which contain details of the author's educational philosophy, as well as teaching tips, and solutions to programming problems.

Both volumes are informal, but very well structured. The main text consists of a short preface, followed by two main sections, seven appendices, references, an index and a table of APL symbols.

The preface explains what APL is; why you should learn it; its strengths and weaknesses; some learning tips; and how to get started. The section on strengths and weaknesses is a model piece of persuasive writing, and should on no account be skipped over.

The first part of the book contains an introduction, followed by nine chapters. This part introduces the APL language; each chapter contains a list of contents, a set of objectives, examples, exercises, a review, and a set of programming projects. The pacing of each chapter has been carefully planned; early sessions go at a comfortable pace, but chapters eight and nine see the introduction of outer and inner product, and matrix divide.

The second part of the book consists of a further five chapters on APL tools. Each chapter describes (and gives the code for) APL programs that are useful in particular disciplines – business, statistics, mathematics, education and computer science. These are working programs that you can use on their own, or incorporate in your own software. The programs are written for clarity rather than efficiency, but they introduce the student to the idea of utility functions – a crucial point, omitted from many courses. Alas, the author does not place equal emphasis on the need to explore the utilities available on the student's own computer system. Utilities can give valuable short cuts in software development, and they can also teach a lot. (If you write optimised but unreadable code, it may be kinder to lock the public versions, for just this reason!) No doubt the emphasis reflects the author's preference for learning by experience rather than example.

At an introductory level, the book compares well with Gilman and Rose. It is even less formal, and much less daunting. It covers less material – if you learn everything in Gilman and Rose, you will know a lot more APL. How many actually do so?

APL – an introduction also adopts some unconventional terms – list rather than vector, input rather than argument. The vexed issue of terminology is discussed in the instructor's notes, along with other contentious topics. I disagree with a number of choices made in this book, but one must welcome any text that emphasises that they are just choices. There are only two such choices with which I feel I must take issue.

The first is on the topic of commenting. In my view, this must be introduced very early on – in immediate execution mode, so students can pick up the habit of first-line comments as soon as they learn how to define a function. Once learnt, the habit sticks for life, but it is much harder to acquire later on!

The second is on the topic of branching methods. Peelle opts for the introduction of branching using the idiom

```
→label×⍴condition
```

This is later superseded by the more normal

```
→condition/label
```

The insuperable objection to the first method is that it goes completely wrong in origin 0. Anyone who learns it will one day waste a great deal of time finding this out the hard way.

These are minor criticisms, however, and the books are generally excellent. APL is easy, fun, and of great practical use. Peelle's new book will help a new generation of students to find this out for themselves.

Handbook and Guide for Comparing Computer Languages

Produced and published by

Research and Education Association, New York (1985)

Reviewed by Peter Branson

After all the discussions on how little publicity APL gets, the thing that hit me when I stumbled across this book in my local library was the front cover. There, in very bold print, are the eight languages covered:

BASIC	PL/1
FORTRAN	APL
PASCAL	ALGOL 60
COBOL	C

It is worth quoting part of the preface:

> "Although most languages can be used for one application or another, a great deal of confusion exists as to which language is appropriate for a given application. From among the various languages that have been developed, eight remain as the most practical and advantageous to use. These are . . ."

and the above are listed again. Although we might not all agree with the chosen eight, at least APL is in there!

After a random dip, followed by a quick serial scan, I realised I was looking at the book in two differing ways: firstly, to see how good were the APL parts; secondly, to help myself with non-APL langauges. (I have a working familiarity with some of these eight, but want to know something about C – of which I am pretty ignorant.) My first impression was that the book was excellent, but this was rapidly modified as I went through the APL sections. In this review I will look at the general parts at the start of the book, and then concentrate on the detailed APL sections, before trying to give an overall summary.

Layout and Introduction

The book starts with a general section on factors for language choice, followed by an overview of each one. The languages are then compared side-by-side under 18 headings, and the book concludes with two sample programs, some appendices and a glossary. The paper, typeface and general layout are very good, and plenty of white space is allowed to get optimum comparison when entries differ in length. The initial section entitled *"Factors for Choice of Language"* is one page covering clarity, simplicity, unity of language and structure, naturalness of application, ease of extension, external support, portability and different measures of efficiency. Although short, this is quite good.

There follows about one page for each language entitled *"Introduction and Brief Overview"*. Within my knowledge of the various languages, the balance seems quite reasonable. The overview of APL is really quite good in what it does cover (which isn't nearly enough – see later), although I do have minor objections to such phrases as "... restriction to homogeneous array structures (creates difficulty in) business data processing", or "subprograms are restricted to at most two arguments (with) a single result", both of which have a stronger negative connotation than is really justified.

APL details

This part of the review concentrates on the APL entries, using the same 18 sub-headings as the book itself.

Program structure:

Good, except there is perhaps insufficient emphasis on the fact that APL is fundamentally different from the other seven.

Statement layout:

Reasonable, although it is not made really clear that a statement separator is only needed when there is more than one statement per line.

Elementary data types:

I don't like this much, although others might think it fair:

"APL is quite restricted in its data-structuring facilities. Numbers and characters are elementary data types."

Identifiers:

This is alright.

Declarations:

Good, although more emphasis could have been given to the fact that declarations are simply unnecessary in APL.

Elementary structured type array:

This is reasonable, but gives a slightly negative impression. Only basic APL is discussed.

Array declarations:

Fine.

Operators:

After quite a reasonable beginning, the bad news now starts. Firstly, someone has clearly decided to call APL primitive functions "operators" (which others have done in the past), presumably to try and get uniformity across the eight languages. However the section then covers only right-to-left execution, parentheses, negation, minus, plus, power, logical comparison, access by name, assignment and indexing. What is done is quite reasonable but woefully inadequate. There is a reference to a "Table of APL Operators", but no such table appears in the book.

Some simple examples are given of the kind:

```
2+3=5
```

but the author does not seem to realise that all the examples will give logical 1 as the result, and the parentheses are missing.

Expressions:

This section is reasonable, except for the use of "infix" for dyadic and "polish prefix" (yes, with a small "p") for monadic.

Assignment:

Quite good, except that only scalar and vector assignments are mentioned, which might cause some confusion since higher-order arrays have previously been introduced (albeit briefly).

Conditional and unconditional branching:

This section is a candidate for the nuthouse. It is frankly appalling and appears to have been written by someone in their first few days with APL. Much of the section is devoted to a laborious exposition of how "times iota" works; this is the only form used anywhere in the book, and often the iota is missing! After all this heavy going, multi-way branches are omitted altogether. With one exception, all branching is to line numbers. There is a reference to the fact that branch to label is "sometimes" useful because of dynamic line number re-allocation, but the example given is poor.

How easy it should have been to include the standard list of "condition, reduce, label" branches, and perhaps the standard IF function as well.

Looping & iteration

Like many of the sections, typographical errors abound. A simple interest calculation is chosen (which doesn't actually need a loop in any of the languages – but never mind that). My APL complaints are that "minus reduce" is used (without explanation) for a simple numeric difference; it may be clever but can only be confusing in this context. Local variables are introduced in a header line without being mentioned anywhere in the book that I have seen. A less important point is the use of a trailing decision.

In particular though, there is no attempt to show how APL can avoid many types of loop altogether; another opportunity sadly missed.

Function (user-defined)

By way of complete contrast, this section is quite well done; even the typographical errors are fewer here than in other sections. It was a heart-felt relief to find four quite reasonable functions, and with branching to labels, no less! The author(s) have clearly battled to try and get common terminology, as noted previously. In earlier sections APL functions are called "programs" or "subprograms", but this section fails to point out that this is what user-defined functions are as well. On the plus side, the terms "dyadic", "monadic" and "niladic" are used correctly, although the attempt to discuss the presence or absence of an explicit result using "unlimited" or "limited" is not so good.

A final point – there are no examples using text or character data either here or elsewhere in the book. This appears to be the case for all the languages and is a notable omission.

Subroutines.

This APL section is brief but good. It correctly describes functions (as "functions"!) and function calling. It also refers properly to unrestricted recursive calls.

Intrinsic/library functions.

In this section the book starts to fall apart again. The text talks variously about "operations", "generator primitives", "expressions", "primitives", etc., whereas what it actually covers are three APL operators – reduction, inner product and outer product. There is further reference to the mythical Table of APL Operators. There is no discussion of library functions, or quad functions for that matter.

Input/output.

A very poor section with no mention of "quote-quad" nor of other forms of I/O available on any reasonable system. There is a passing reference to files which are "not normally provided", but at least "some recent APL implementations include such features".

Program halts.

The entire entry is "Not applicable". The authors have presumably never heard of "quad delay", or of function-defined halts, etc.

Documentation.

The final subsection deals with comment statements within a function in a moderately acceptable way, although the example given is poor.

Sample programs.

Two sample programs are included and the best word to describe them is "hideous". Riddled with typographical errors, as usual, but also with such charming features as:

numbered header lines;
all branches use "times iota" to line numbers;
two "quad" entries for data input, rather than a vector;
etc.

The examples under user-defined functions could not be called brilliant, except perhaps in comparison with these.

Appendices.

I. Pascal delimiter words (1 page)
II. ANSI Cobol reserved words (2 pages)
III. Summary of Cobol formats (4 pages)
IV. PL/1 built-in functions (1 page)
V. C syntax summary (2 pages)
VI. PL/1 Arithmetic built-in functions

Without checking a reference manual, I believe that IV and VI are both PL/1, so perhaps one of these should be the missing APL table.

Glossary.

An excellent glossary comes last, although it is far too extensive for the subject matter of the book. It includes gates, bytes, buffers, cascaded arrays, clock pulse, etc. – terms which don't appear at all in the text. Some of the 25 pages used (out of a total of only 122) could have been put to better use.

Well that's the end of the book as such, but I can't leave without listing some of the more glaring omissions. Workspaces and system commands are mentioned but only in the Overview. The following are not covered at all:

many of the important functions;
local and global variables;
quad functions and variables;
quote-quad and other I/O forms;
text or character data;
library workspaces and functions;
public domain software; draft ISO standard;
portability; fast prototyping;
modularity;
access to other languages and systems;
availability on micros;
nested arrays;
user-defined operators;
other features of enhanced APLs.

Before summing up, I will paraphrase some of the claims made on the back cover:

- Enables comparison of all eight languages at a glance.
- Makes for rapid selection of the most appropriate language.
- Transition between the eight languages made easy.
- Enables design of completely new languages.

Leaving aside the last, rather grandiose, claim, the very good design and style of the book, should have made the other claims possible. However the appalling errors and omissions in the APL partly nullify these claims, certainly as far as APL is concerned. I imagine a potential APL user would be totally confused.

As for my other interest – can it help me quickly to get a grasp of C? Well it appears to give me something of the flavour of the language, but naturally I am left wondering how many errors there are in the C entries. (A cursory look at the Fortran and Basic parts suggests that these are not too bad, but I still wouldn't trust them without a critical review.) So unless someone will kindly review the C parts for me, it looks as though I am going to have to fork out for *Kernigan and Ritchie* after all, and I suggest that anyone wishing to learn APL sticks to *Gilman and Rose*.

My overall sentiment is one of intense disappointment; such a good idea was ruined in the execution. The book is well worth a look (if only to see what might have been), but I have not been able to track down a U.K. supplier; in any case I would recommend your local library, rather than spending well-earned money on this edition. A second edition, thoroughly checked for accuracy and completeness would be well worth having, and I am waiting to hear from the Association in New York whether or not one is planned.

Building on a solid past

1976
ISAM/AP for APLSV
VSAM/AP for APLSV
1977
AFM/AP® APLSV
AFM/AP for VSPC/VSAPL
1978
AFM/AP for CMS/VSAPL
CALL/AP for APLSV
1979
CALL/AP for VSPC/VSAPL
1980
AFM/AP for TSO/VSAPL
1981
CALL/AP for CMS/VSAPL
Enhanced Format for VSAPL
Keyed Access Option
1982
Interactive Link Option
Mail Exchange Option
CALL/AP for TSO/VSAPL
1983
APLPRINT
Input Stack Processor
Output Stack Processor
1985
AFM/AP for CMS/APL2
AFM/AP for TSO/APL2
CALL/AP for APL2
Enhanced Format for APL2
1986
APL2 Display Capture
Indexed File Option

When Interprocess Systems got started ten years ago, they had a lot of ideas about APL.

As far back as APLSV, they believed that APL users needed better file facilities. They began by providing shared access to ISAM and VSAM.

Then came a component file system that would run on any in-house IBM mainframe APL system.

But their customers wanted more than just component files. They asked for - and got - keyed access and an indexed, PCS-style file structure.

There was also a need for a Quad-FMT equivalent that wouldn't change IBM source code. Interprocess did it for VSAPL, and it's still there for APL2.

Recently APL2 added the ability to call FORTRAN or BAL from APL - something that Interprocess introduced as long ago as 1978.

Interprocess also pioneered a concept: That APL enhancements don't have to change the APL product itself. That you can achieve results from the IBM interfaces already present.

It's a concept that gives Interprocess customers unparalleled continuity from one APL environment to the next. Something solid to build on for the future.

For more details of the IBM APL enhancements developed by Interprocess Systems contact

APL Software Limited
27 Downs Way, Epsom, Surrey KT18 5LU
England
Tel. 03727 21282

Product reviews

APL.68000 for the Atari ST

Reviewed by Paul Chapman

Preface

The following review assumes some experience of the APL language and environment, and some of the concepts, such as workspaces, common to all APL installations. It also assumes at least a passing acquaintance with WIMP (windows, icons, mouse, pull down menus) environments.

Introduction

The machine provided by MicroAPL was an Atari 520ST with high resolution monochrome screen and two disk drives. The software provided for review was sealed and obviously an end user production version. A full set of software and documentation for the 520ST was also provided.

The Atari 520ST and its younger, bigger brother the 1040ST are 8Mhz 68000 based micro-computers. The ROM-based operating system is called TOS. An Apple Macintosh-style desktop environment called GEM, which is driven by a two-button mouse (included as standard) is provided, also in ROM. On disk, Basic and Logo language interpreters are provided, as well as a word-processor called 1st word.

I concentrated my attention on the APL interpreter, and, in particular, its interface to the GEM WIMP environment.

Getting Started

The interpreter comes in a box proclaiming "APL.68000 PROFESSIONAL PROGRAMMING LANGUAGE" and costs £170 plus VAT. The single disk provided contains the interpreter, five utility workspaces, and a demo workspace. The documentation comprises a generic APL.68000 language manual, which is provided with all implementations of the interpreter; a much smaller manual *APL.68000 for the Atari ST* containing a description of the desktop environment, and details of the workspaces supplied; a reference card; and a copy of the latest issue of *MicroAPL News*.

I had no problem getting the interpreter up and running. No special installation procedure was necessary – this is one piece of sotware which you can take home, plug in, and start using straight away. APL key stickers were already affixed to the keyboard supplied, and a set was also provided with the package. These would probably take about half an hour to put on.

My first problem came when I tried to load the DEMO workspace using the menu provided. With the mouse, I selected the "File" menu from the top of the screen, and then the "Open" item from that menu. After a little grinding from the disk, a list of the workspaces on the APL disk appeared in a new window, and I selected "DEMO" by pointing to it and "double-clicking" on it. (This piece of WIMP jargon indicates pressing the select button on the mouse twice in quick succession, which has the effect of opening the

selected item.) The new window disappeared, and after a while the message "WS LOCKED" appeared in the main dialogue window. Repeating the attempt produced the same effect.

A manual)LOAD was successful, and thereafter the problem of being unable to load workspaces from the menu disappeared. Furthur investigation revealed that the problem only occured the first time after the interpreter was loaded, and before any expression had been entered. It's an annoyance rather than a bug, and would be especially so to a newcomer, who might spend some time in the manuals trying to find out what he or she is doing wrong, or to find some other way (in this case,)LOAD) to get started.

The demo workspace illustrates very simply how APL can be used to write applications which use the WIMP environment and graphics on the ST. Typing DEMO caused the menu bar at the top of the screen to change to a list of menus defined in the application.

From the menu, it was possible to call up one of two graphic displays (a rather boring graph in one, and a number of variously shaded and patterned squares in the other), and also a "Dialog Box" which allows the further selection of options and switches (in this case a mock-up of a very simple serial port set up). The final menu item allowed me to return to native APL, either with or without clearing the workspace.

I began examining the functions in the workspace. The DEL editor worked much as expected, and used the normal APL dialogue window. Also provided is a full screen editor, which can be entered from the menu using the mouse. Upon selecting "Open fn" from the "Edit" menu, a dialogue box appears to ask the name of the function. The default is the last function edited, although perhaps a more appropriate default would be the function appearing at the top of the SI.

The editor uses the whole of the dialogue window, first of all saving the current contents, and then displaying the function selected on the screen. The editor is natural to use – I didn't have to look in the manual to learn fancy control characters. My main reservation was that there was no distinction of lines beginning with labels or comments. This, taken together with the fact that blank lines, although they can be entered, are not retained in the function definition, made reading functions rather difficult.

I was impressed with how little code was apparently needed to interface to GEM, especially after some unpleasant experiences with the Commodore Amiga system documentation, which weighs several pounds and takes weeks to comprehend. The use of the graphics utility functions was immediately self-evident, and even the calls to GEM itself were fairly transparent.

Some of the functions, for example those used to set up menus and dialogues, took global variables as arguments. Unfortunately, the full-screen editor could not be used to examine and edit character vectors and matrices, which was an irritation.

The global arrays themselves were again mostly self-explanatory. For example, the array used to define the application menus was a character matrix, with menu titles starting in the first column, with the menu items for each title appearing below the title, on lines beginning with one space.

Typing a few APL expressions supplied sensible answers, though the overhead in display time produced from working within a window environment is slightly irritating after the memory-mapped characters of the IBM PC, particularly when scrolling. This shouldn't bother 3278 users, however!

A simple interactive graphics function

I now set myself the task of writing a function which would allow very simple line drawings to be produced under mouse control. The idea was to be able to point the mouse at the screen where I wanted the line to begin, then press the left mouse button and hold it down while I "dragged out" (another piece of WIMP jargon) a dotted line until I was happy with its position, then release the botton whereupon a solid line would be added to the picture.

I went to the pamphlet describing the particular features of the ST version of the interpreter, and quickly found a function called GETMOUSE in workspace TOOLS supplied with the system. This niladic function returns a 3-element vector of the current x and y co-ordinates of the mouse pointer, together with an integer from 0 to 3 indicating the state of the mouse buttons.

In the description of the STGRAPH workspace, I found a number of tools for line drawing, shape drawing and shape filling functions. LINECOLOR sets the line color, while LINETYPE sets the line style (solid and various patterns of dotted). Finally, POLYLINE draws a line or sequence of lines. The funcion CLEARWINDOW clears the window of APL dialogue, which may be restored with RESTORESCREEN.

It took twenty minutes of trial and error to set up this function, which worked quite successfully, if a little sluggishly.

Many problems and some solutions

Many problems came to light during this process, however. Many could be dealt with by a careful study of the descriptions of the functions, but some were deficiencies in the implementation itself.

In particular, an irritating lack of elegance and consistency came to light in the use of the graphic functions.

The origin for mouse co-ordinates is the top left corner of the screen, whereas that for all graphic co-ordinates is the top left corner of the current window. Whenever it is necessary to translate from one to the other, an offset must be supplied which is supplied by the function WINDOWPOS. This can be overcome by taking the additional steps of setting a global variable called WHOLESCREEN to 1, and then calling CLIPRECT with argument WINDOWPOS.

This must be done repeatedly, since it is possible for the application user to move the window and change its size. Unfortunately, if the user does this more than once, the interpreter restores the contents of the APL dialogue to the window, wiping out any output from the application so far.

Some of the graphic objects are expressed using (x,y) co-ordinates, whilst the rest use row-column co-ordinates (ie (y,x) co-ordinates). This may be a reflection of the way the internal GEM routines work, but inelegance in the way GEM works should not be passed on to the APL programmer.

None of the graphics drawing parameters (eg. current line color, line type) can be interrogated by the programmer: the usual convention, that of making functions return the old values when given an empty argument, is omitted.

Most graphics systems allow different interactions to be specified between graphic objects to be drawn and the current contents of the screen. For example, it is usually possible to define XOR or inverting interaction, so that two successive placings of an object on the screen will have no net effect. This was essential in the simple program described above, for the purposes of drawing the "dragged out" dotted line which changes position as the mouse pointer is moved about the screen.

The graphics functions supplied allow this to be done only with solid objects like circles and rectangles (and not with their outlines), and then only some interactions are possible, eg. not AND or OR. No interactions are possible with line drawing. Eventually I chose to drag out a solid rectangle, which was ugly and unclear, but at least it was possible to do this in inverted display.

Presumably, the GEM kernel itself allows all these functions, so I can't see any problem in providing them to the APL programmer.

Another problem which emerged was)COPYing from a nearly full or full disk. For some reason,)COPY uses temporary disk space, so the message I/O ERROR – DISK IS FULL can come as something of a surprise during a read operation.

There are all sorts of nasty little problems associated with using the cursor to select extracts for cut/copy/paste operations. For example, it is not possible to select a full line for pasting in the full screen editor.

It as also impossible to get in or out of the full screen editor without using the mouse, which is irritating. Also, the mouse pointer disappears altogether during output to the screen, so that if one moves the mouse to the menu bar during output, the position of the pointer cannot be seen.

Documentation

I did not need to look at the APL language manual once, which is encouraging. What I was using was certainly a real APL, and worked fine.

The rather thin pamphlet describing the implementation is rather unsatisfactory. For example, the default colour numbers are described as 0 for black and 1 for white. In fact, they the the reverse of this. The descriptions of the utility functions are far from formal, and occasionally misleading because APL programmers should naturally assume consistent behaviour, since so much is made of this in the APL language itself.

Software

I did not have time to explore the other workspaces provided. These are: MENUS for creating menus and interrogating the user's selection of items; DIALOG for creating and executing dialogues; STFILE for accessing ST native files; and TOOLS which contains miscellaneous functions for programming function keys, setting keyboard translation, etc.

It really looks to me as if the STGRAPH workspace was cobbled together with the rest in a hurry to create a product. It also looks as if little or no beta testing of the product has been performed. I set out to review the usefulness of the implementation rather than to try and break the interpreter, and yet I found several bugs, inconsistencies and irritations within a very short time.

The functions I explored are simple to pick up and use, but this is more a result of lack of flexibility than careful and logical design. Much of GEM is inaccessible from APL, except through the use of custom-written shared variable processors and careful study of the GEM system manuals. If MicroAPL were to consider expanding the range of functions for accessing GEM in the future, they would find consistent extensions to the functions already defined very difficult.

Conclusions

APL.68000 is a sound product as an APL interpreter, and competes fairly well in functionality and very well in price with its main rival on micros, STSC's APL*PLUS/PC.

The ST is used both at home and in business. Much of its appeal over, say, the IBM PC must be in its price for home users, and in the GEM environment for business. It is difficult to see which market APL.68000 is aimed at. Its price is too high, compared with that of the hardware it is running on, to be attractive to home users, and the lack of a full interface to GEM would put business users off.

For myself, if I owned an Atari ST, I would spend my money on a C compiler so that I could have full access to GEM's facilities. If I had to recommend the purchase to a business user committed to the ST, I would point out the need to spend money on having a custom interface produced by a systems programmer if any serious use of the GEM environment were planned.

APL is praised for its natural extension to graphics environments, but MicroAPL have a long way to go to provide the access to GEM an APL graphics programmer needs.

STSC APL*PLUS/PC release 6

by Martyn Adams

At last I have received a copy of STSC APL*PLUS release 6.0 for the PC! I always look forward to receiving a new version of my favourite piece of software. This version's great claim to progress is its ability to handle objects over the 64k-byte limitation but there are several other features which are really quite useful.

We quickly discovered that Grade-up and Grade-down give NONCE ERRORs when trying to process objects larger than 64k. So you can't sort large objects. We also found that ⎕VI and ⎕FI give DOMAIN ERRORs on large objects, so you can't validate large character arrays either.

We also tried the ASMFNS workspace and found that some of the assembler functions were slower than 'doing it by hand' in APL. I haven't checked but I suspect they may have used less workspace. STSC, could we have some ASMFNS written in 80286 machine code (for our ATs) for even faster code?

The 64k-byte variable size barrier has never actually been a problem to me personally. If any object went over that size I guessed that something had gone wrong with my coding. If you only have 400k or so of workspace on a 640k machine then 64k can be a very large percentage of space devoted to one object. WS FULL has always been my problem. Nevertheless a dark cloud has been removed and if you want to process a large object then, provided you have the workspace, you do not need to chop it up any more.

This begs the question... when can we break the 640k machine size barrier? Already lots of software take advantage of extra memory. Sometimes this memory is banked so as to appear as another parallel memory space. Sometimes it sits on top of the operating system (DOS) as extended memory. I prefer the latter idea – it is easier for me to understand.

Anyway, we all wait for the latest version of DOS running on IBM's new PC based on the Intel 80386 chip. This should break all known speed and size barriers which currently limit the PC architecture. It is rumoured to be called DOS 5.0 – we all wait with anticipation.

Meanwhile COMPAQ have beaten IBM by releasing their COMPAQ 386 to the public. It runs very, very fast. It is to IBM PC/AT what the AT was to the IBM PC/XT. COMPAQ seem to marketing the new machine as basically a go-faster DOS box. It runs a version of MS-DOS and does one or two tricks with its internals in order to maximise performance. And it is claimed to be pretty nearly 100% compatible with IBM's PCs.

Unfortunately the latter statement isn't strictly true. I had a chance to run APL*PLUS release 5.0 on it. Everything seemed to work fine except that the inner product always returned 0. Even when the APL statement should have returned 1. This was very disheartening. I suspect that there may be one or two other areas which do not give results as expected. I understand that release 6 will also not run on the 386 so we will have to wait for a 386 compatible APL.

⎕WIN, which handles the PC screen as a full-screen manager, has been improved. The few improvements are very useful and can be quite significant if you program them properly. Unfortunately the new manual that comes with release 6, although nicer than the old one, is still a little confusing when it comes to describing the actions and effects of the keyboard and what characters are allowed where under what circumstances. After a little experimenting though, all becomes clear and it really isn't as difficult as it first seems.

Function keys are now handled better, so they say, but I still find the definition of a complex data entry/edit screen a long and tiresome process.

STSC have improved the way APL starts up in "noddy" mode for the first time users. I haven't explored these features as I consider myself a bit of a power user but I understand that for the first time user these can be really helpful.

Other nice features include ⎕CHDIR, ⎕RMDIR, ⎕MKDIR and ⎕LIBD. The first three mimic the DOS commands for changing, removing and creating directories. These are very useful. Especially useful is ⎕LIBD which allows you to dynamically set up and delete (and ⎕LIBS interrogates) the APL library structure. Before ⎕LIBD you had to use a file called APLLIBS on start-up which is a very clumsy method of defining libraries.

I do however have a gripe with the native file naming convention. In this release you can define a native file called: 'B:DATA\FILEA.XXX' and it is found, as one would expect, on the directory called DATA on diskette drive B:. This is much better than the old method where it was very difficult to read files which lived in directories other than the current one.

Full marks for that one – but if you specify a native file called: 'B:FILEB.XXX' then the APL will not look for the file where you would expect. The DOS standard is that the file should be found on drive B: in its current directory. However the APL replaces the drive B: characters with the library 1 definition (drive A: would be library 0, drive C: library 2 etc.).

This, I thought, was confusing. 'B:' means drive B: if there is a backslash in the file name; otherwise it means the diskette drive and directory specified in library 1 (except on alternate Thursdays?). I suppose the developers at STSC know what they are doing. I don't.

⎕GPRINT has been upgraded to handle the LaserJet Plus printer – I haven't had time to try this option but am looking forward to it. STSC have thoughtfully supplied a little reference manual along with the revamped documentation. Called the *Quick Reference Guide* it happens to be so full of information that it is not a very quick reference at all (it is quicker and more handy than the manual though). It is so useful that I hide it from my colleagues.

In addition to the comprehensive language description the *Quick Reference Guide* gives details of the 87 odd different authorised ⎕PEEK and ⎕POKES. Additional POKES include:

- whether or not you wish to allow low minus as negative signs in character strings when validating them using ⎕VI or converting them using ⎕FI.
- Inhibit creation of objects greater than 64k.
- Allow the display of sub-directories and/or volume ID when using ⎕LIB.
- Disable buffering of keyboard input while functions are executing when using the DOS keyboard routines.
- Make ⎕TRACE give a short form output.
- Force a return to text mode from graphics mode whenever immediate execution is entered.

Incidentally the *Quick Reference Guide* doesn't specify all the parameters for ⎕GPRINT.

Finally, as a general point, I would just like to add that as an APL purist at heart I am disappointed that there are now 144 ⎕functions in APL*PLUS release 6. I guess we may soon be able to forget the APL and write in ⎕code only. I also feel uneasy about the way that a lot of machine-coded functions (using ⎕CALL) are supplied as optional extras and one, BOX (which turns a character vector into a matrix), is practically compulsory.

The fact that APL*PLUS has to use so many ⎕functions and the odd machine-coded program indicates to me that perhaps the APL should be made a little more practical. It should understand about its working environment a little more, even if it means that variables and files are one and the same thing! I do not blame APL*PLUS for their policy of enhancing their APL by the addition of ⎕functions (in fact it is a very rich language for this reason) – but somehow I feel that something is being lost from the concept of APL as a

computer programming language. I guess it has something to do with the fact that APL is really a computer implementation of Iverson's notation. Still, full-screen management and full file management as APL primitives have their appeal.

In conclusion then, APL*PLUS release 6 is a definite evolutionary step forward. Well worth looking at for the serious PC APL developer. I consider it a real professionals' tool and an improvement on release 5. A lot of barriers have been lifted – roll-on release 7.

APL.68000 for the Apple Macintosh

by M.S. Bassett

The Macintosh was the first readily available computer to use the concepts of mice, windows and pull-down menus to build a user interface, and whatever you think of this approach yourself much of the strength of the Macintosh is that nearly all of the software you run on it will offer the same working environment and access methods.

This poses interesting problems for the writers of Macintosh programming languages; they must provide:

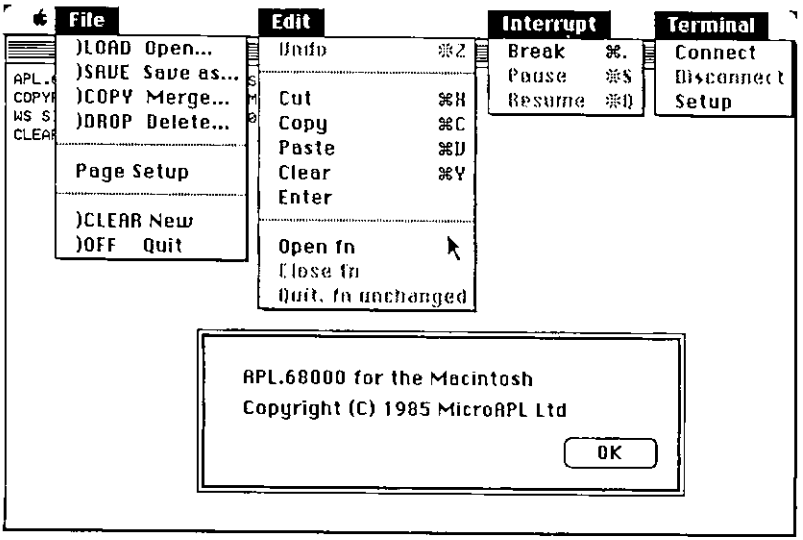
a) a Macintosh-style interface for the potential programmer (who is, after all, a user as well);

b) a respectable programming language;

c) a means for that programmer to provide a Macintosh-style interface for the users of their program in turn. This review will attempt to describe how MicroAPL have fared in each of these three areas.

The APL.68000 interpreter is in fact very comfortable to use, and integrates nicely into the Macintosh environment. To start the interpreter you may either use the mouse to point to the interpreter itself, which loads and gives you a CLEAR workspace, or point to a workspace file on disc which will invoke the interpreter and automatically load the workspace selected. (Macintosh owners note: obviously you don't just point to these things you double-click on them, but I want to avoid as much MacJargon, sorry Macintosh jargon, as possible in this review.)

When loaded, the interpreter offers you a 24 by 80 text window, which can be moved about the screen but not re-sized. At the top you have a choice of four menus which control the APL environment, including the editing and running of functions (see figure 1). I particularly like the File menu which offers you both the APL and the Macintosh descriptions of workspace loading/copying etc.



When you select an option from the File menu a window appears asking you to select the name of the workspace desired from a menu (the menu scrolls if you have more than eight workspaces on the disc). You also have the option to cancel the request or change disc drives of you wish. This is all just like the standard Macintosh approach but I found two oddities: first whether you are loading, copying, merging or deleting, the text on the window says "open" i.e. load a workspace – this led me to accidentally delete on of the workspaces provided with the system; second I could find no way to save a workspace of the same name as one already existing on the disc – very frustrating !

Editing a function is done via the Edit menu (surprise !). The edit window is large and scrolls in all four directions, again in the standard Macintosh manner. The APL character font is small but perfectly readable. Editing is peformed using the Macintosh "cut and paste" approach which is essentially a set of highly flexible block move/copy/delete operations. This is also standard Macintosh style but I found it a bit awkward; I kept having to take my hands off the keyboard to use the mouse and in particular it is very frustrating not being able to move the cursor from the keyboard. My normal style of APL programming is to write a line of code from left to right leaving gaps or pairs of parentheses for bits I am going to come back and fill in later; this involves a lot of cursor work and is especially unsuited to use of a mouse. However after half an hour's practice I had written some fairly long functions so perhaps familiarity will bring content.

There is no recognised "Break" key on the Macintosh keyboard so MicroAPL chose the cloverleaf key for this role which seems a reasonable choice; confirmed mousers can choose Break or Pause from the Interrupt menu which will normally be available whenever a program is running.

Unfortunately I had no facilities with which to try out the Terminal emulation menu, so all I can report is that it offers the standard VT52 protocol and a wide variety of option selections.

Moving onto the interpreter itself, the language is a full implementation of APL.68000 which is jolly good news; what everybody wants to see of course is the benchmarks. The first few columns of this table were cribbed from VECTOR Volume 1, Number 4.

UPDATED 'SMITH' BENCHMARKS

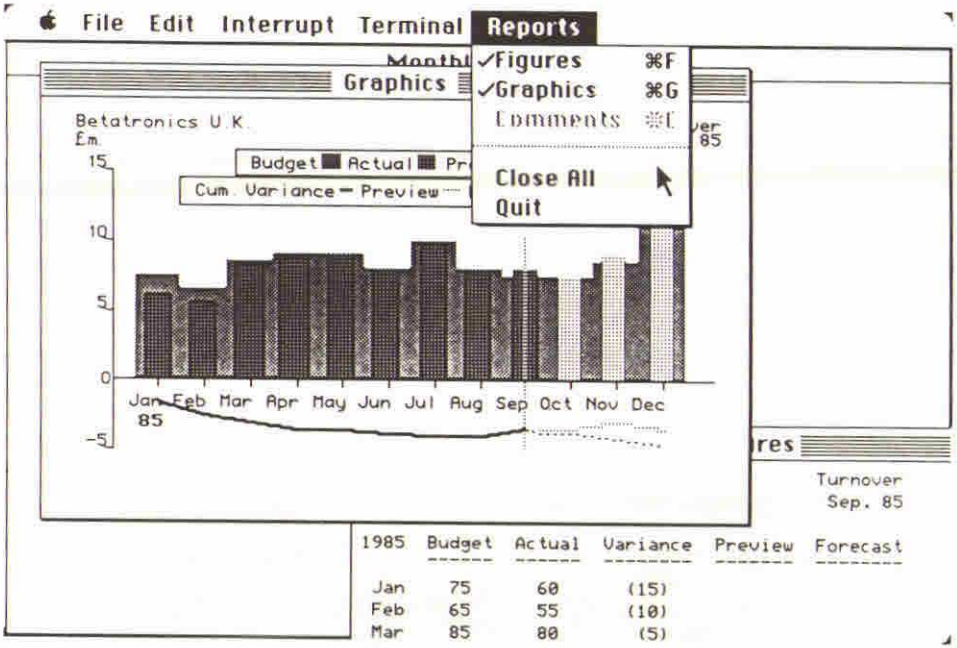
	IBM PC	*IBM PC*	*SPECTRUM*	*QL*	*MACINTOSH*
CHIP:	8088	8088	68000	68008	68000
APL:	*IBM APL*	*APL**PLUS*	*APL.68000*	*QL/APL(K)*	*APL.68000*
VERSION:	1.00	2.6	4.10/B	1.03	1.0
BENCHMARKS:					
Z←+/*VI*	90	102	26	54	18
Z←∨/*VL*	0.4	3	6	16	2
Z←⌊/[1]*MI*	40	25	20	42	11
Z←*VI*×.1	390	282	4312	6824	3494
Z←|*VR*	80	79	87	169	64
Z←*VR*[*VI*[⍳20]]	20	14	23	51	12
Z←*VI*[⍒*VI*]	600	112	190	325	150
Z←¯2 1↑*MR*	9	24	12	27	4
Z←*VI*∈*VI*	150	146	217	202	100
Z←2 1⊗*MC*	450	60	188	355	141
Z←*VC*∘.=*VC*	360	141	158	363	117
Z←(⍳50)∘.+⍳50	2530	439	241	535	165
Z←*VR*⌊.+*VR*	210	341	192	340	148
Z←*MR*⌸10↑*VR*	70	1488	340	559	270
Z←*FIB*	2200	3827	1654	3250	1255
Z←*VR*×3.14	100	136	229	396	180
Z←*VR*⋆3.14	110	142	389	541	306
Z←⍟*VR*	150	143	2428	3807	1968
Z←10○*VR*×.1	411	438	3462	4559	2256

As you can see it is very fast !

The last hurdle to clear is making use of the Macintosh graphics facilities from within APL, the "Toolbox" of special purpose routines in the Macintosh ROM. MicroAPL provide a selection of several workspaces to this end, each containing a set of assembler routines accessible from APL to control such things as menus, the mouse, graphics windows etc. I was unable to find any aspect of the Macintosh interface that I couldn't control with these routines and they are very reasonably sized, taking up 60K of a 272K workspace if you load everything in.

Using these routines you can create your own menus and menu bars, over-riding the standard options if you wish, track the movement of the mouse, make windows pop up all over the place and generally dazzle users with any amount of pyrotechnics. You also have full access to the standard Macintosh data-transfer facilities, the "Scrapbook" and "Clipboard". The graphics routines are particularly impressive. Standard functions let you create lines, circles, ovals ,boxes and arbitrary polygons, filled or not with whatever pattern you desire; and these functions are in themselves quite fast. However by using the MAKEPICTURE command the output from these functions can be coded into an integer vector (which I presume is a set of calls to the ROM); in itself quite small this vector can be processed to re-draw the original image at blinding speed. As a small example of what can be achieved I offer figure 3, which shows my favourite "half-hour" database with graphics, customised menus and a choice of output windows, the entire screen taking perhaps two seconds to draw.

In short this must be the best APL currently available for the Macintosh and quite likely one of the best available for any PC. Highly recommended.



RECENT MEETINGS

This section of VECTOR is intended to document the seminars delivered at recent meetings of the Association, particularly for the benefit of those members based away from London who often find it hard to find the time to attend. It also covers other selected events which are likely to be of interest to the wider APL community.

We are dependent on the willingness of speakers to provide us with a written version of their talk, and we would remind them that "a picture's worth a thousand words". Copies of slides and transparencies will enhance their articles.

The Activities Officer (see inside back cover) will respond enthusiastically to offers from individuals who wish to contribute seminars and supporting papers.

British APL Association Meeting

I-APL

17 October 1986

Reviewed by Anthony Camacho

I-APL had been advertised as the only topic. The plan of the afternoon was for Anthony to explain what the project was about, then for Norman Thomson to talk about what is being done to prepare for introducing it into schools and finally for David Ziemann and Paul Chapman to answer any questions about the specification and how the interpreter is to be produced and ported to the target machines.

The explanation of the project's history and content took the form of five large charts. The main events so far, the reasons for the project, the target specification, the details of the features of the interpreter (ISO conformance) and the organisation of the project.

Chart 1: I-APL Main events so far.

April 1986	First discussed
APL 86 July	Committee formed 30 enthusiasts
August 1986	SigAPL votes $5000 (conditionally)
September 1986	Letters to all European Groups BAA votes £6000 (conditionally)
October 1986	Specification in draft Marketing plan to be produced Work on interpreter begins
November 1986	Fundraising begins in earnest
.....	
APL 87 Dallas May	Interpreter on demonstration

It was made clear that the project is not under the auspices of the British APL Association or SigAPL and that it should not be assumed that what the members of the project said expressed the opinions of the BAA. The details of the conditions under which the sums voted by SigAPL and the BAA would be paid were made clear. SigAPL wishes to see and approve of the marketing plan (as it is not a commercial product some things will be harder and a few easier than the usual kind of marketing), and will pay $500 on evidence that the project is really going to start, $2500 on approval of the marketing plan and $2000 on completion of the interpreter (to assist with the costs of launching it and distribution). The BAA conditions are that its Technical Officer vets the Technical Specification and reports

his acceptance to the committee and that its Publicity Officer and Education Officer are given the opportunity to see and suggest improvements to the marketing plan, and that the project accepts any amendments they require.

When asked what the state of the BAA finances was, the BAA Officers present agreed that they could not really speak for the Treasurer in his absence; however they could safely say that the BAA had more than the proposed contribution in its current bank account and that this was before the profit from APL 86 was taken into account; in short all that could be said was that it would not strain the BAA's finances to make this contribution.

Chart 2: I-APL – Reasons for the Project

An interpreter which will run on small computers is needed to give more people the chance to try APL (especially at school and at home).

A free interpreter is needed to overcome the price barrier which prevents many people from taking their interest further (most schools cannot afford commercial prices).

The target machines for the project are the BBC 'B', Spectrum 48K (and up), the Apple II (and up), all CP/M micros (e.g. Amstrad word processors) and all PC look-alikes (e.g. Amstrad 1512 series).

The major task besides producing the interpreter itself will be to provide introductory material suitable for this large potential readership: teaching material for schools, games for home use and manuals and guides which do not assume the degree of determination to absorb new ideas that a company expects from a data processing employee.

The purpose of the whole project is to promote the use of APL: not specially I-APL, which will neither be fast nor well provided with facilities, but any APL. If the project succeeds it will bring many new members to the BAA.

(Editor: As the I-APL specification is printed in full in the technical section of this issue of VECTOR, we have omitted charts 3 and 4 which were an abbreviated specification.)

David Ziemann explained that although it may take some time for the formalities to be completed before the ISO Standard is issued, the draft is now fixed and we can safely assume that it will not change.

Paul Chapman explained that the interpreter will be designed to be ported easily onto any new machine and the source code will be freely available so that the project does not have to do all the ports – or even to have knowledge of all of them.

Paul Chapman explained that this outline specification is a target not an absolute commitment (some facilities may have to be dropped if there is no room for them), but that he intended to write the interpreter with all these facilities built-in and only to drop features as a last resort if he could find no other way of getting the interpreter into the space.

David Ziemann offered to provide a copy of the draft specification as it stood (not yet agreed) to anyone with a legitimate interest, provided that recipients would treat it with discretion and not allow it to be published until agreed.

Chart 5: I-APL: Project Organisation

The Committee is:

Edward M. Cherlin (Editor: APL Market News) joint chairman
Anthony Camacho (Sec: BAA) joint chairman
Howard Peelle (Prof. of Education, Univ. of Massachusetts)
Norman Thomson (Educn. Officer, BAA – algorithms ed: Quote-quad)
David Ziemann (Technical Officer, BAA)

The Committee has formed a Company called I-APL Limited and all committee members are Directors. Anthony Camacho is Company Secretary.

There are at present nearly 40 enthusiasts on the mailing list, mostly collected at APL 86. Anyone interested may add their name to the list and will be circulated with news of progress.

David Ziemann explained that he hoped to improve the draft specification as a result of comments made at this meeting.

It was agreed that it is most important to make clear that all manuals and other documents issued as part of the project could be copied and translated freely (the latter point especially important to users of APL whose native tongue is not English).

Norman Thomson pointed out that as far as the bulk of the new potential users were concerned, the project will not have a product until the supporting documentation is complete and can be provided with the interpreter.

The documentation at present being prepared consists of a Tutorial on which Linda Alvord is working, an encyclopaedia of APL (giving examples of each use for the primitive functions, operators and system features in alphabetical order) which Garry Helzer is writing and a series of books or booklets to be called "APL for . . . ." where the dots are replaced by a particular application of APL.

The obvious first example was *APL Programs for Mathematics* and Norman circulated draft copies of his book with that title (90 pages of A4 text). This was tried out by looking up in it some school mathematics topics such as Pythagorean triples and going through the examples given. Norman will welcome comments from those who took copies away.

This book (and Garry Helzer's, which is also in draft) at present has only standard APL examples: eventually there will be side by side examples both in APL and in the ASCII mnemonic transliteration. This will make the book suitable for use whether or not the computer has an APL character set available and will help to accustom people to the APL characters at the same time. Llewellyn Jones has been working on a way to do this transliteration for some two years now and is willing to give the project permission to use his work free of charge. He has functions which will translate from either form into the other under VS APL and under APL*PLUS/PC. (APLpip for the PC and APLpup for the mainframe where EBCDIC is a bit more restrictive than ASCII.)

Norman said that the marketing effort could only begin when both interpreter and documentation were ready. He suggested several lines of attack. As these were discussed and added to during the meeting the set below includes all the main ideas that seemed acceptable to those present.

1 Prestige schools. Norman Thomson has contacts at several of these and believes that as they have better facilities and lively and open-minded staff, they should be easier to persuade to try out APL. If two or three of the best known schools can be shown to have made a success with APL then it may be easier to overcome resistance in other schools. Excellent reference sites will certainly do us no harm.

2 County Education advisers (or Inspectors as they are called in ILEA – which Neil Bibby pointed out) should certainly be made aware of what we are offering and asked to encourage their schools to try experiments with APL.

3 British APL Association members should be asked to demonstrate what we produce to anyone in education that they know and to help to give away the interpreter, sell any books we have to charge for and provide the project with feedback on what other material could be provided which would help to make the product more acceptable.

4 Plainly any school which can be persuaded to give APL a try should be; the selection will have to be haphazard – it will be the teachers members of the APL Association or the I-APL Project know rather than any selection by some criterion of suitability.

5 It was agreed that it would be particularly valuable to get I-APL exposed to the trainees and staff of teacher training colleges. A few committed advocates in such an institution could encourage the spread of APL out of all proportion to their numbers.

6 It was suggested that another good place for publicity was the user groups. There are groups interested in particular machines, microcomputer clubs for particular districts or in particular colleges and these enthusiasts are more open to new ideas than the general run of home and school microcomputer users. It might even be possible to get the interpreter put onto a public access network so that people could download it.

Norman Thomson reported that the Mathematical Association has a book giving the BASIC programs for 132 common mathematical functions or processes and that this is very popular and widely used. His book is intended to upstage this as it contains many more functions (about twice as many) and need not be more expensive. He has already been exploring ways to get it published as cheaply as possible and is weighing the relative advantages of cheapness and getting onto the lists of a recognised and respected educational publisher.

Les Hollingbery asked about style and standards of the examples in the material to be produced for the project (he liked the style of Norman's examples). It is desirable that the examples should encourage good habits but the project ought not to try to impose 'good' style on potential users (at least not where there is room for a difference of opinion about what style is 'good'). The project cannot really afford an editor to impose a house style on authors who are all doing voluntary work and may be unwilling to accept anyone else's views of how to do it. Any suggestions which could be circulated for comment to all authors working on the project would be very welcome.

When the meeting was thrown open to general discussion (questions had been encouraged throughout) many helpful suggestions were made. The dash between the I and APL could subtly be made heart-shaped. We should write and include some games in the free software issued with the project. We should all write to computer and non-computer press about the project to give it publicity and arouse expectations; perhaps we could ask magazine readers to send in their names and addresses if they were interested in an early copy of the interpreter.

Paul Chapman was asked about the process of writing the interpreter and he explained that he intended to write code which would be interpreted in every machine. The interpreter would be a very simple version of a threaded language – something after the style of FORTH but not FORTH itself because the interpreter would be too big and complex. The APL interpreter could thus be exactly the same sequence of ones and zeros on every machine. It would use the machine code entirely through the kernel of the threaded interpreter. To port the APL to a new machine it would be necessary to write the machine language kernel for the machine and code the calls to the operating system which did such things as write a character to the screen, get a character from the keyboard and so on. As it happens only three machine language kernels will be required – to suit 6502, Z80 and the 8088 family. Paul hopes that his development environment will have commercial value. It is to be called DE (to imply it is better than C as an environment for creating portable software) – he was going to call it DEL (for development environment language) until he discovered he was unaccountably losing files!

The general atmosphere of the meeting was friendly and no-one voiced any adverse criticism of the work that had been done so far or of the plans presented. When the audience was challenged to suggest a better way to encourage the spread of APL no suggestion was offered. When people were asked explicitly whether they thought the contribution from the BAA was a good use of the funds there was no dissenting voice to say that it was not. There was general agreement that this was a very good way to try to spread the use of APL, and many present felt that so far unfruitful efforts might meet with success once they had the proposed free product to help them.

Thirteen people put themselves down on the list of people interested in having news about the project and most of them offered to help in some way.

The meeting ended at about 5.30 and discussions continued in the bar for some time.

APL In Action

Manchester
England

7-11 July
1986

APL 86

acm

The British
Computer
Society

APL Debate: What is APL Thinking?

Reported by David Preedy

Howard Peelle, who acted as chairman for the session, introduced the debate by outlining why he felt that the topic of APL Thinking was an important issue to discuss at the conference:

> "It's a term that we seem to bandy about quite freely here in the APL community and if indeed we are serious about the dissemination of APL, especially for those people who are learning APL for the first time, it seems to us that it is important to understand APL thinking ourselves."

He then said a few words on his own perspective on the subject by way of warm-up, describing some of the work done by himself and his colleagues at the University of Massachusetts, addressing the objective stated by Robert Hooker:

> "The time is right for the psychology of programming and of APL in particular to come of age . . . we have a need to study the unique mechanisms used by APL programmers in structuring their thoughts".

Their approach to their initial studies into APL thinking has been to study the literature and extract relevant comments; to analyse some common mistakes that beginners make when learning APL; and to think about the kinds of errors that APL instructors make while teaching APL. With this background they've begun a series of interviews and published a survey in Quote-quad inviting people to say what they think APL thinking means to them.

They have initiated a course daringly entitled "APL thinking" – a computer science course attracting also some people from education and from mathematics. As a result they have identified quite a number of challenges or related issues surrounding the topic; they concern questions such as the extent to which individual style affects APL thinking or, as he phrased it:

> "Do you believe that APL thinking style or APL programming style can or should be taught?".

Despite its inherent difficulties, they are using a methodology involving asking people to talk out loud while they're thinking about a problem and while they're using APL to solve it.

Peelle finished off his introduction by showing a few quotes that they have extracted either from the literature or from the individuals directly about what people think constitutes APL thinking.

> "Clear, simple rules; compact notation; unambiguous interpretation; executability of mathematical statements"
>
> "APL thinking is recognising patterns, decomposing a problem and seeing where to apply APL idioms"
>
> "I know I can do it iteratively but the question is how can I do it with arrays, be it elegant or not".

"The best way to gain fluency in APL is by thinking in APL"

"It is not evident that the average programmer thinks in arrays".

Several quotes explored the ideas of visualising geometric objects and dealing with mathematical expressions.

"Problem-solving using APL can frequently be facilitated through the use of visual imagineering"

"I learned APL simply by considering how a mathematician would think".

"APL is the essence of mathematical thinking"

or, by the same speaker when pressed on the point:

"Mathematical thinking is the essence of APL thinking".

"APL is a good short-hand mathematics"

"APL thinking means ecstacy"

"It's the ability to say the same thing in several ways"

Some quotes explored the importance of modelling – structuring data rather than program flow; thinking about the problem rather than thinking about the program one's writing or certainly rather than thinking about the machine upon which it's going to run.

"It involves being the black sheep in the D.P. department"

"One could memorise all the APL primitives and other aspects of the language and still not be able to solve problems"

(APL thinking is what would be missing in that case.)

"There's a large distance between how I think of a problem and how I must write it in APL".

APL thinking is not APL itself; it's not what APL does; or even what APL makes easy to do. It's what we do beyond what APL offers. It may include what we have to do to make APL fit the problem or even make the problem fit APL. Therefore APL thinking is really the complement to APL. The last quote was:

"APL thinking – I don't know what that is".

The first panellist on the debate was Micheal Berry:

Having been involved with teams working on designing parallel computers, Berry presented the view from one who has recently moved outside the APL world to a place where people are spending a lot of time doing exactly the opposite of what those in APL think people usually ought to do, namely thinking about the details of a particular computer architecture and how to write programs that will match the way the machine works. The machine in question is a massively parallel machine called the Connection Machine which involves somewhat over 65000 processors all of which are fairly small but have their own memory and each of which has the sole option of whether or not to execute the the instruction that is sent to it from a host computer.

The problem with how to program such a machine therefore is how to design programs (and how to design a programming language in which to design those programs) in which it's easy to make statements so that each of the little processors working independently of the others will usefully progress towards solving the problem.

> "One of the first weeks I was there, people were sitting around discussing the problem especially with the IF statement for the new language. The problem being that... you imagine arrays as being stored with one element in each processor and, if you had 2 arrays that you were going to add, the corresponding elements of each array would be in different memory locations but on the same set of processors and you would send the instruction ADD presumably with some pointers to the 2 areas in memory and each little processor would do its one addition, and so the thousands of ADDs would be done.
>
> "Well, they knew they needed to design an IF statement and the obvious way for it to work would be that you evaluate what they call the predicate, this being a LISP environment, basically a Boolean expression, and for all the processors in which it's true they stay turned on and do the next thing and then they turn themselves off, and the other set of processors turns itself on and does the ELSE thing.
>
> "People were worried about the fact that unfortunately this decision-making has to happen in the front-end computer, something like a VAX or a Symbolics 3600, and a lot of time is wasted while thinking about doing the IF while the Connection Machine itself is sitting there not doing anything. I brought up an idea which seemed weird to people: 'Why don't you try and write the programs so there aren't any IFs, because you would think of an instruction to say, which would do the right thing to all of the data. The instruction would be more complicated but each processor would do it and have done the right thing.' "
>
> "The reason I thought of that and everyone else was thinking about how to do the IF was because I had just come from years and years of thinking in APL and it seemed natural to me to try and think of a solution where even if it's more complicated to express what you're doing you can still do it all at once."

They now regard suitable problems for the Connection Machine, as those exhibiting a high degree of "data-level parallelism", meaning that you can do it all at once, such as image-processing, where each pixel can be instructed to ask its neighbours how bright they are and average with them or something like that.

Berry saw this as one angle on what is APL thinking; it's thinking about how to find that data-level parallelism, because data is what we can structure very well in APL. He illustrated this concept with some examples.

The first was taken from a *Scientific American* article some time ago which compared computer languages by asking a skilled writer in each language to write what they think would be the normal way in their language to express the problem "Add up all the odd integers in this integer vector". The APL solution differed radically from all the other solutions in incorporating data-level parallelism; all the other solutions involved looking at each element and asking if it were odd and if so incrementing the counter variable that's collecting the sum, and if not going on to the next one and asking it. So in fact Scientific American had found a good example of APL thinking.

Another familiar example is the problem of rotating the lines of a text matrix to remove leading blanks. In APL, you work out how many leading blanks there are in each line and rotate by that amount; you don't loop through the lines. Whilst each line does have to move by a different amount, you only need one expression by which to figure out how much they have to move.

Sometimes it's hard to see the data-level parallelism because at the level that you're looking at it the data doesn't look quite parallel or looks ragged and here Berry showed how nested arrays have helped in letting parallelism or rectangularity be imposed at whatever level is most convenient.

Berry finished by exploring the trade-off between elegance and performance. One of the things that restricts us from really practising our APL thinking is that people look at you and say "That's cute, but come on let's be reasonable it's not an efficient way to do things". At Analogic, Berry had great fun implementing the paragraphing system using domino and realising that, while still perhaps slower than some other ways, it worked fast enough for him to use it, and he did use it in his text editor just for fun. If Berry does achieve his goal of getting a "thinking machine", and if he succeeds as a mole-bearer and we ever have APL on the Connection machine, he remains confident it will be better to use the approaches currrently discarded on grounds of performance. He looks forward to the day when his APL thinking will not only be clear to himself and hopefully to readers, but will also be the best way to make the hardware do what it's supposed to do.

In answer to a question, Berry explained how he felt that the Connection machine has been affected by APL. The proposed specification for the language for the Connection Machine borrows exclusively and fairly heavily from APL where the most noticeable things are operators reduction and scan. Although not called operators they they apply to a function and give you a new function that works just like APL ones. Another thing, that would also be an operator by our APL definitions for scalar extension, applies to a function and makes a function that is executed in every processor (in Connection Machine terms); in APL we would say it applies to each element of the array, so it's either a scalar extension operator or an "each" operator of some kind. These were quite conscious borrowings and Berry found that in joining the Connection Machine team his knowledge of APL was recognised as a plus.

The next speaker was Ray Polivka. He started by exploring some of the premises underlying the debate. The first was that the individual concerned actually is prepared to think, and wants to think. It was an illusion to forget that many people simply do not want to think about the process they are learning.

His next statement was:

> "I don't think there is APL thinking.
>
> "Let me say it a little differently – I believe APL fosters thinking and the type of thought patterns that it fosters are structure driven, or data driven. It allows us, if we so choose to do it, to think more like the patterns that we would like to think in. Now I'm comparing that with the patterns that are forced upon us in regular computing."

Based on his experience in teaching APL to a whole spectrum of people from engineers, secretaries, managers and computer science trained people, Polivka had found that the problems lie with the computer science folk, because APL allows you to think more naturally.

> "What is the natural way that the computer scientist has been trained in? IF THEN ELSE I GETS INCREMENT etc DO WHILE FOR. That's why what has happened in many cases is that the person who has come from a PASCAL trained background comes into APL and proceeds to write glorious Pascal with APL terminology."

One of the other problems is that programmers tend to drop down too quickly to the details. If you're going to build a piece of furniture, you don't immediately go down to a hardware store and pick out the nails you're going to use. You design the thing first – what shape and size you want – negotiate with your wife or husband and things of this sort. You don't go down to the details.

And yet in fact in programming in a sequential language you are forced to go down to those details much earlier than you want to. APL promotes thinking which is:

> Algorithmic,
>
> Idiomatic – although Polivka preferred a term from psychology "chunks of APL" because you can read the statement item by item or function by function or you can see it as a chunk of APL
>
> Inquisitive – the fact that APL is executable on a machine is a great bonus and if you're not sure what happens, you just try it out.
>
> "Multi-perspective" – meaning that there's no one way to the solution. One of the real joys of working with APL is to come to a collleague and say "Look at the solution I have got to this problem" and he can say "Look at mine" and you find that you've been looking at it from two different ways.

Polivka closed by saying that APL is a notation that we need, will continue to need in the growth of our scientific and engineering development as a society and culture. APL fosters thinking.

—

The next speaker was Roy Sykes, who started by pointing out the inadequacy of the normal adjectives used to describe APL thinking – words like "modular", "array-driven", "whizz-bang", "intuitive", "parallel" and so forth. He thought those terms are inadequate. He prefered a metaphor – the metaphor of the combined architect and builder of a house, somewhat like Polivka's furniture builder. The client, often oneself, is interviewed; the needs are assessed; the blueprints are drawn and approved; the required materials are acquired; the proper tools are brought to bear; the house is constructed; and finally some walls are shifted around, a few changes are made to better suit the client. Sykes saw APL thinking covering this broad context. It is not limited to what one does with the symbols of APL. It's the mental process that threads throughout the entire process of defining and solving the problem.

Sykes illustrated this by going through the typical process that he typically undergoes in defining a program, a function of a modest size, not a throw-away one but one that is part of a larger system.

> "The first thing I do is think; think about what the problem is, try and understand what I'm doing. Part of that process is understanding exactly what my inputs are. This is nothing new; one decides whether one is taking character or numeric data, is it going to come in as arguments or global variables from the user at a keyboard, from files, whatever. But in any case I define that very carefully. At this point I open definition on the function, and I have the formal header, perhaps not the local variables, and perhaps 3 or 4 lines of comments describing precisely what my inputs are."

Next he does the same with the outputs – decide what the outputs are, precisely how they are structured and ordered. He has now documented a substantial part of the function without writing any code – presuming that the header is not code.

Then he thinks about the transformations required; whether they're structural or mathematical primarily; is it inherently a parallel or an iterative solution? Of course that is overlaid with what we can currently do in APL with the primitives available to us. This involves deciding whether to adopt a nested or a simple solution. Essentially he breaks the problem into sentence-sized chunks.

The next thing is what some other people would consider APL thinking and coding – Sykes overlays the sentence-sized chunks onto the tools and mechanisms he knows. Working in order, he starts with the APL primitives; for a structural problem he may think in terms of taking transpose, drop and laminate; for a mathematical problem he may consider the scalar primitives, ⌹divide, base-value and so on. Then he introduces the so-called idioms or chunks that are available that we all know. Thirdly he brings in the existing commonly used subroutines he has already used to solve problems. And fourth he thinks in terms of new subroutines that he might write and how they could be generalised for future applications.

The penultimate step is to lash all these chunks together trying to smooth the transition between the sentences, putting in the appropriate commas, semi-colons, periods, paragraphing, and so on – the nails of Ray Polivka's furniture.

Then Sykes tests his code. The testing has three phases. First he assures that the proper inputs result in the proper outputs, especially on the edge conditions. Secondly he checks that bad inputs are handled properly, giving correct responses to the users, or signalling errors to the outer environment or just letting the program blow up – that's acceptable provided that the error performance is documented. Third is testing no undesirable outputs arise, such as unlocalised local variables and so on. We don't after all want the neighbour's house subsiding!

Finally, the last step is incorporating revisions. He reviews the core algorithm used and removes any redundancies. The chunks have a bit of a disadvantage in that if you simply write in chunks, you may end up with a chunky program. One wants to have a smooth program, and sometimes if you look at the broader measure you see that two or three chunks put together are rather an entirely new algorithm. Sykes removes these redundancies and revises the documentation.

> "If we define APL thinking in its very narrowest sense, that of the transformations and coding, we encourage a myopic approach to problems. I teach APL and I find that the most difficult problem is not APL thinking, it's thinking. The results are fuzzy because the inputs and outputs are not even clarified by the students to start. He or she has a very broad idea of what they want to do; they don't know know really what they start with, and they don't know really what they want. Once those two things are clarified, the process is simplified by APL. If we can teach people to clarify their thoughts, the embodiment of their thoughts into APL code is simplified."

In response to a question, Sykes explained his approach when he finds he actually gets stuck in defining the algorithm. If none of his colleagues can help he puts together the very simplest, often iterative, scalar solution to solve the problem.

> "I get something that works; it may put my machine to sleep for 30 seconds but at least it works. It represents the most basic, (pun intended), solution to the problem. It works, I know my inputs, I know my outputs, and the transformation there. That process will often clarify to me what I have to do in a more parallel sense, using the APL primitives at hand. Very often, especially these days, I find myself at first thinking in terms of nested array solutions, and then backing off into simpler solutions that are almost as terse but run considerably faster using simple arrays."

—

The final panellist was Norman Thomson. He felt that his view of APL thinking might reflect a transatlantic difference in attitude from his three co-panellists. To him APL thinking is a kind of layer that goes on outside APL; it's the kind of things that he says to himself in order to make APL make sense.

At an earlier session Stephen Jaffe had formalised little rules – rules that are the baby-talk of APL – the things that make it easy to get it right if you are re-structuring multi-dimensional arrays. What you do is you build up a fabric of these informal rules, informal as opposed to the formal rules of APL, and that forms the basis of Thomson's APL thinking.

He then looked at the increasing challenge presented by APL2:

> "Now, I guess that if we had reviewed this five or six years' ago, before APL2 came widely onto the scene, we could see that we'd to a large extent exhausted the APL1 challenge – we'd begun to understand and largely control the symbols that are present in APL1. In my bookcase at home, I've got a book of silly ideas. I once had the idea that it would be fun to take all the APL primitive symbols, and combine them, take all the possible pairs of symbols and work out what they did together; maybe take a few triads and find out what groups of three did; explore the possibilities and see what you got, and hopefully quite a lot of them might be interesting. In a sense when you've done that, and you've filled that book, then you would know all there was to know about APL, that's the end of it and I could then go on and study something interesting like beetles or mushrooms or something of that sort. I hasten to say that it's a book with a lot of empty pages, but there was a point at which APL1 looked like a closed universe."

Thomson then gave an illustration of how his existing APL1 informal rules had been affected by the introduction of APL2. He looked at the case of outer product. Intuitively there he had an interpretation; it meant extending an argument and applying a function to any string of numbers to get blocks of results. It only becomes meaningful, to do an outer product in code, when we had the "each" operator, and so it was lovely when APL2 came along and proposed "each", because somehow it was the jigsaw-piece that neatly filled in a gap that was somehow a void in APL1.

However the other side to this particular coin had come to mind earlier in the day when an expression involving nested arrays had been displayed and Thomson had fallen hook, line and sinker for an incorrect interpretation. He had been forced to go back to first principles and even then his interpretation was rejected in discussions with others. It seems that with APL2 we have created a structure of vastly greater complexity than we ever had with APL1. Some time ago APL1 looked like being a nicely closed, nicely rounded, self-contained unity. Now with APL2, the "clear, simple rules" and "unambiguous interpretation", so admired by Howard Peelle's respondents seem to have evaporated. As Thomson concluded:

> "Yes it's unambiguous, but goodness me it takes a lot of seeing to perceive that unambiguous interpretation."
>
> "So in short that's my perception of APL thinking – something that has radically changed between the, what now seems relatively simple, structure of APL1 and this vast relatively unexplored territory of APL2."

—

Following the formal presentations by each of the panellists, there followed a more general discussion involving members of the audience as well. There was considerable discussion on the alternative interpretations of some APL2 idioms and whether they could be read naturally left to right, read aloud for instance so that a class can understand them.

From the floor, Anthony Camacho called on the principle of Occam's Razor, saying that it seemed to him that the types of thinking identified as APL thinking do not appear to differ from ordinary thinking. So he was inclined to agree with Ray Polivka when he said that there isn't APL thinking – there's just thinking. The opposing view was given that APL thinking is in fact a subset of ordinary APL, and the debate had been concerned with identifying the specific characteristics that are typical of APL thinking in particular.

In response to Norman Thomson's APL2 problems, it was hypothesised that he had given an example of is how APL thinking is something that he has developed. When he looked at the expression and applied APL thought to it, he didn't get the right answer, because in APL2 APL thinking doesn't work!

Adin Falcoff suggested that the answer to Anthony Camacho's dilemma was that in discussing APL Thinking we are working in the context of the world of programming and so when he talks about APL thinking he is contrasting it with thinking about other programming languages.

> "I think there are some very simple and obvious differences. You tend to think in terms of transformations of arrays; you tend to think in terms of functions that have arguments and results; you tend and you learn eventually to think in terms of operators; and the consequence of this is that you learn to think of breaking your problem down in some logical way. Some of these things are present in other programming languages, but most of them are not, and certainly the others are present to a much larger extent in APL than in other langauges."

Ken Iverson looked at the different types of approach to describing processes, or even to describe the same process from different points of view. Sometimes one wants to emphasize iteration, sometimes recursion, and there's also the use of arrays. Then there's the question of doing things modularly and having functions with arguments and results; on the other hand also possibly deciding things in detail. He thought that the essential thing of any good language is that it allows you to express yourself in any of these ways conveniently and cleanly and to the extent that APL is successful it is because it does that. Iverson finished with a question:

> "How much harm do you think that the people in APL have been doing and are doing by emphasising that APL thinking has one narrow notion that of arrays?"

Roy Sykes believed that we have been doing a lot of harm and asked how many times we have wasted time trying to make things faster, in particular by making them non-looping, when the first solution that comes to mind is an iterative solution. He felt that one of the biggest boons of an APL compiler is that it allows the freedom to think in ways that either are not overlaid on the primitives and operators that we have at hand, or not intuitively overlaid on them.

Norman Thomson pointed out as evidence of the distinction between APL thinking and ordinary thinking that in his own personal experience, APL had affected his life in some sense – he saw things differently once he had been shown APL. He also disagreed with Adin Falcoff's view that the context of APL Thinking had to be programming. He felt that the one thing that characterizes APL and makes it different from other languages is that its context is a great deal broader than just programming. The revelation about APL was its relevance in the real world; that, for example, the encode function is what you do when you get change; or it's what you do when you convert from centimetres to Imperial units.

> "It seems to me that the whole essence of APL, and the whole thing that keeps conferences like this going year after year, is the fact that APL has a spark, a bit of inspiration, a bit of something I don't know what – the thing we're trying to identify, I guess – that is just that much broader than pure programming."

Adin Falcoff responded that he thought that people's interest in it would be somewhat diminished if it were not implemented on computers – that programming should not be regarded as a dirty word. Norman Thomson's reply was that APL was the thing that to him at least humanizes computers.

Linda Alvord explained how, as a mathematics teacher, she has a mental model allowing her to see the structure of APL as if it was the 3-dimensional world that we live in. There's a visual aspect to the images of the data that is quite different from the way you think about it in other programming languages. There's something about the thought process that involves image-making, and model-making, that characterizes it as different from other languages.

Ray Polivka felt that we need to encourage array-thinking. He paraphrased what Anthony Camacho had said at the Education Day:

> "Loop-thinking is not natural; more and more programmers are learning more and more on how to be baffled by parallel array thinking."

APL isn't the only language capable of handling collections of data – vectors, arrays, strings, etc. – but one of its other strengths is it provides the supporting function is to manipulate these things very concisely and precisely.

Roy Sykes finished the debate by highlighting APL's interactive executability as fundamental to its nature. But primarily, he thought:

> "APL is a new vocabulary and it's been said, by whom I don't know, that without vocabulary, without language, thought is impossible. Language in some respects defines thought. And the language that we're speaking of, which is APL, defines our thoughts in a much broader way, a much richer language, than any other computer language that I know of. I think the very fact that we have such a large vocabulary in APL makes our thinking process larger and richer in solving problems."

LOWEST PRICES!
DIRECT FROM THE U.S.A.!

Hundreds more
available — call for a quote!

HOW MUCH WILL YOU SAVE?

call now! **(01) 997-4277**

800 SERVICE 800

FREE CALL!

USA DIRECT SOFTWARE
(301) 762 6647

Telex: 9102400147

1377 K Street, N.W., Suite 827 Washington, D.C. 20005 USA

Idioms and Problem Solving in APL2

Delivered by Alan Graham at APL86

Transcribed by John Sullivan

(Editor: The start of the tape is unfortunately inaudible.)

. . . A very common one (idiom) I call 'All Right' takes each item on the left and pairs it with the entire array R on the right. I have some examples which will make this clear. To take the whole array on the right, if that's what you want, you don't want it itemwise.

```
L f¨ (⊂R)
```

'All Left' is the whole array on the left.

```
(⊂L) f¨ R
```

Now what's the final combination of EACH and ENCLOSE? How about if you enclose both of them? Not interesting or important, because you have that identity.

```
(⊂L) f¨ (⊂R)  ↔  ⊂ L f¨ R
```

So there's 'All Right', 'All Left', but 'All Both' doesn't make sense.

One thing you find when using EACH is that the derived function with RESHAPE is a scalar function.

What that means is that like addition and subtraction items get paired and, if you have a scalar, scalar extension occurs; dealing with scalars you can think of it as the RESHAPE to the size of the non-scalar and then the function is applied pairwise. This is important because RESHAPE is not a scalar function, but RESHAPE EACH is a scalar function. FOO EACH is a scalar function – I don't care what FOO does, it's a scalar function. I found that hard to adapt to, once I'd adapted to it I say, as you do once you've learned something, "Of course It's simple", but you don't say that while trying to adapt to it because it's tough.

OK we'll look at some examples and try and adapt to it. A two-element vector plus a two-element vector, you get a two-element vector, same thing here,

```
2 3⍴4 5
```

that's 2 rehape 4, 3 reshape 5. Well, how about if I'd like for instance down here I'd like two 2 by 3 matrices, one filled with 4s, one filled with 5s. That's it, I want to use the entire left argument over and over again, so I enclose it, you get scalar extension

```
(⊂2 3)⍴¨4 5
```

so it's the same as writing (2 3)(2 3) both in parentheses RESHAPE EACH and then carry out

```
(2 3⍴4),(2 3⍴5)
```

And the final one, 'All Right'

```
2 3⍴¨⊂4 5
```

I get enclosed vector 4 5, and enclosed vector 4 5 4. And this occurs time after time after time. I guess – er – well I'll leave it at that.

I want to show you some other applications of this, this in action. Garth Foster called me up, well, I guess I called him, a few weeks ago; he had a question that after a little head-scratching I came out with, I knew the answer, I don't know why it took me so long. He said you have the alphabet and you'd like to map indices into the alphabet into words, and more specifically, he wanted to get a nested array of indices representing an array of words, and he'd like to do that mapping and extract the words. So, here's one reason why I use index-origin zero, so you can put the blank in front and A is one, and blank is zero. Here is my alphabet

```
A←' ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

and with PICK it works easily, you can take out the I like that

```
      9⊃A
I
```

and during one organized conference they had the competition for what words and phrases in history mean. Before this talk, because there is the CHIPMUNK idiom, they had it after this talk everyone in this room wanted to win that prize. Why is it called the CHIPMUNK idiom?

```
9 2 13⊃¨⊂A
```

You see, doesn't that look like a chipmunk to you? Eyes, and big fat cheeks there. So there's the chipmunk idiom for you. It has to be asked, which one of the templates is it, pairwise, all left or all right; which one am I using? (Pause, then answer from the audience 'All Right') All right. I'm taking the entire alphabet, and picking from it the ninth, then picking from it the second, then picking from it the thirteenth and I get IBM out of it and, well, hold on, what is the F; the F is PICK EACH, that's the scalar function that I call F. So that's just a case of All Right.

Now, what Garth wanted to do – Oh, so you could say it was an index I'll call I, and then you could use the chipmunk again

```
      I←9 2 13
      I⊃¨⊂A
IBM
```

and then you could say, well I want to add one to each number and compare that to the addition of zero,

```
     II←(⊂I)-0 I
     II⊃¨¨⊂⊂A
IBM HAL
```

but then of course I'd get IBM and HAL from 2001! This is what Garth wanted to do, and this is another example of All Right, because PICK EACH ENCLOSE (the chipmunk idiom) is the chipmunk function, you could call it, so I'd like to do a list on the left, that function, EACH, All Right; so I have this idiom which really would have been an obscure question. What do you think that idiom is called? You'll never guess so I'll tell you. That's called 'chipmunk with glasses and a toothache'. (Laughter) You'll have to use it a whole bit more and have great fun.

When I invite people in to see these things or give demos I tell them that this is my APL2 PC, even though it's on the mainframe, and these are my APL2 toys, although I think they're not as trivial as toys. Another way to do this if you want to avoid the pepper effect, you can define a sub-function called vector indexing, which is very nice.

```
[0]  Z←I VI V
[1] ⍝ VECTOR INDEX
[2]  Z←I⊃¨⊂V
```

now you can say 'jot-dot-VI', you can say 'VI-each', you can say 'VI-each-each-each-each', and you can use the function without thinking 'chipmunk', which might not have much to do with vector indexing. This works fine, (inaudible), and now it becomes apparent that we're using All Right here, because here is the function, a list of indices or a list of lists of indices, and All Right on the right.

```
      DS II
┌→──────────────────┐
│ ┌→─────┐ ┌→─────┐ │
│ │9 2 13│ │8 1 12│ │
│ └~─────┘ └~─────┘ │
└~──────────────────┘
      DS II VI¨⊂A
┌→────────────┐
│ ┌→──┐ ┌→──┐ │
│ │IBM│ │HAL│ │
│ └───┘ └───┘ │
└~────────────┘
```

Now lets have some fun with EACH. Take a look at that.

```
      X←(2ρ¨3 5 7)ρ¨⊂I,0
      DS X
┌→──────────────────────────────────────────────────┐
│ ┌→─────┐ ┌→──────────┐ ┌→──────────────────┐ │
│ │9 2 13│ │9  2 13  0 │ │9  2 13  0  9  2 13│ │
│ │0 9  2│ │2 13  0  9 │ │0  9  2 13  0  9  2│ │
│ │13 0 9│ │13  0  9 2 │ │13  0  9  2 13  0  9│ │
│ └~─────┘ │0  9 2 13 0│ │2 13  0  9  2 13  0│ │
│          │9  2 13  0 │ │9  2 13  0  9  2 13│ │
│          └~──────────┘ │0  9  2 13  0  9  2│ │
│                        │13  0  9  2 13  0  9│ │
│                        └~──────────────────┘ │
└~──────────────────────────────────────────────┘
```

Well, let's see. Only two EACHes. 'Two reshape each' you get 3 3 (pause) 5 5 (pause) 7 7. We say we want to apply each one of those to the entire thing to the right, which are the indices I followed by zero, so we want to do 3 3 reshape of that whole list, then a 5 5 reshape of that whole list, and a 7 7 reshape of that whole list, and we get this beautiful thing.

I feel that with APL1 I've been pedalling a very very nice bicycle, and with APL2 they gave me a motor (some chuckles from the audience). It's very easy to get horribly carried away, and you should maybe restrain yourself; so I can use the chipmunk with glasses to solve Garth's problem of mapping arrays of indices into characters

```
      DS X⊃¨¨⊂⊂A
.+-------------------------.
| .+--. .+----. .+------. |
| +IBM| +IBM I| +IBM IBM| |
| | IB| |BM IB| | IBM IB| |
| |M I| |M IBM| |M IBM I| |
| '---' | IBM | |BM IBM | |
|       |IBM I| |IBM IBM| |
|       '-----' | IBM IB| |
|               |M IBM I| |
|               '-------' |
'-------------------------'
```

A little bit different example, of how EACH helps you out, and how derived functions and operators can help you out. I just made this array that I'd like to manipulate, and you might make that array just experimenting: it's a 3-element vector, and whatever you can do: a scalar, a vector and a matrix. If you display it it looks like this

```
      DS R←3.14 'ABRACADABRA' (2 3⍴4 3 2 1)
.←----------------------------.
|       .←-----------. .←----. |
| 3.14  |ABRACADABRA| ↓4 3 2| |
|       '-----------' |1 4 3| |
|                     '~----' |
'-----------------------------'
```

You take the shape and, sure enough, it is a three-element vector

```
      ⍴R
3
```

Sometimes I ask my students a trick question, "How many elements in a 3-element vector?" They get that (laughter). "How many elements in a scalar?" One, we know that, but sometimes – hmmm – zero? "What's the shape of a scalar?" One? No, so the edge conditions are tricky; this is a normal 3-element vector, nothing up my sleeve. You also may want to say, what's the shape of each one of those items

```
  ⍴¨R
11  2 3
```

and you can clearly see the first item is empty, which is correct (It is indented, it's hard to seewhen you get a wider display font like this), an 11-element vector and a 2 by 3. In the spacing of the original system this is pretty close to it, there's two blanks there, and so you can only default this place, you do get the feel for the separation here if you look carefully. I find I don't use this display too terribly often because I do like to default this place. One of the other things I tried is – Ah yes, of course, I want the rank of each one, zero one two

```
      ⍴⍴¨R
3
```

That doesn't work. Why? Anybody know? I mean, three? Where did that come from? (answer from audience RHO-EACH) Rho-each is a scalar function, what's a scalar function on a 3-element vector? A three-element vector: that's not a trick question. So what's the shape of a 3-element vector? Three. Even my students get that, very easily. So, of course I tried this, thinking "Ah, I'll fix it". (laughter)

```
      (⍴⍴)¨R
SYNTAX ERROR
      (⍴⍴)¨R
        ^^
```

and dialled up Jim Brown and said "Hey, it doesn't work". Well, sure, we could make this work, but I think rho-rho – I think you mean rho-rho made-up in a direct-definition form, but you could mean W-rho-rho-W or you could mean – there's many different functions so we didn't choose one: no, that doesn't work. The thing is not a function, so you get SYNTAX ERROR. Well, you can do it rho-each-rho-each

```
      ⍴⍤⍴⍤R
0 1 2
```

and here's some pepper occurring. You'll also notice the spacing here, there's two spaces between the items, so if I'm doing rank-each, I know the rank of anything is a single number and I prefer the simplest data-structure possible so now we really get the scrambled eggs with pepper and you get zero-one-two and that's exactly the right answer I want.

```
      ⍴⍤⍴⍤R
0 1 2
```

Hmmm, what can I do. Well, I take stock and write this rank function,

```
      ⎕FX 'Z←RANK X' 'Z←⍴⍴X'
RANK
```

and that I find the easiest way without popping into an editor, just define on the fly, and then you RANK-EACH

```
      RANK⍤R
0 1 2
```

I find that what I want is not to have to stop to define functions then to come back to all my problems but I never have the function RANK lying around in all my workspaces because it's just too easy, so what do I do? (How am I doing on time? OK? What's that? Fifteen minutes, ah, OK means different things to different people (laughter) I can only give one percent of my talk now (more laughter).) I want an operator, an operator to the rescue. My friend Phil Benkard calls this idea a 'Nonce function' (Nonce means 'for the moment') Some APL systems give out NONCE ERROR when they're not sure what should be implemented or not. IBM says "We can't do that" because nonce is pre-announcing something that's going to be . . . . OK. I call these nonce functions so I can have a function for the moment so I can apply each to it, or outer product or what-have-you, and then have it evaporate.

Here, what I do is I use the simple form of direct definition what I mean is I don't do the if-then-else case, sure you can do it, but I have another way to do if-then-else and what I do is I take a function in a character representation form, a character string. Operators always produce real live functions so anywhere in APL2 where we say you can use a function, you can use a primitive or a defined function or a derived function. So therefore what I get out of this operator taking a data operand, which I feel a little queasy about, is a derived function. I don't feel so bad because the F represents a function as character. So let's go through it.

```
[0]  Z¯←Y¯ (F¯ DD) X¯
[1] ⍝ Direct Definition (APL2's Lambda)
[2]  ⎕ES(2≠⎕NC'F¯')/5 4      ⍝ Variable?
[3]  ⎕ES(1<⍴⍴F¯)/5 2         ⍝ Vector/Scalar?
[4]  ⎕ES(~F¯≡⍕F¯)/5 4      ⍝ String
[5]  (('ω'=F¯)/F¯)←c' X¯ '   ⍝ Replace ω←right
[6]  F¯←∈F¯                  ⍝ Simple
[7]  (('α'=F¯)/F¯)←c' Y¯ '   ⍝ Replace α←left
[8]  F¯←∈F¯                  ⍝ Simple
[9]  '⎕ES ⎕ET' ⎕EA 'Z¯←',F¯ ⍝ Do it under trap
```

What I do is I do the error checking first, as primitives must do error checking of their input before they try it out, and I'll read it in English as fast as I can. F must be a variable coming in, not a function because this doesn't apply. Then I say if it's over rank 1 it's no good, it's got to be a string, then I say check – if it's not a string blow up. I remember that 5 4 is a domain error, but I got it wrong – I put in 5 1 and was getting VALENCE ERROR. Well, I sit with the book next to my desk, or I could type it in quickly to see what error I get. Right here I say if there are any omega's in this string replace them with the name of the argument – oh, I should mention, why the little marks next to each one of the names? I'm trying, in a vain attempt perhaps, to avoid name conflict, because the thing you're executing may reference variables, global variables, or may call subfunctions – if you have a subfunction by the name of X, or worse F, this doesn't work If I had not put the suffix of an overbar; now things won't work if you have a subfunction F-overbar, but I'm using that convention to say "these names I really want to be strictly local" APL doesn't give you the facility to say "make them strictly local", so I do it by naming convention.

So finally I say take this square peg and fit it into the round hole because this is 4 characters long and I'm replacing occurrences of scalars, so I enclose it, get rid of the nesting, and enclose the other argument. Now I never do a name-class to see if this function is monadic or dyadic if it's monadic there won't beany alphas in there I don't even have to check. The last statement says execute the right argument and if it fails for any reason take the error you got which is stored in ⎕ ET and record it one level higher because I may have a function whose argument I pass is not in the domain, the operator has no idea of what's right, it blindly applies functions, so this is very common in operators, to say do it now and oh by the way if you blow up don't report it here, report it one level up. Just as you don't get the assembler language code for outer product, if you do an outer product jot-dot-divide and you have some zeros lying around, I don't want the code for my operators to show up in the same way. So, how can I use this? There's my original array

```
      R
3.14 ABRACADABRA    4 3 2
                    1 4 3
```

I would like to say the rank of each item, conveniently, and I say it here

```
      '⍴⍴ω' DD¯ R
0 1 2
```

I'd like to say de-duplicate each one: give me the nub of each item,

```
      '((⍵⍳⍵)=⍳⍴⍵)/⍵←,⍵' DD¨ R
3.14  ABRCD  4 3 2 1
```

and it works. Also, you can be the judge, sometimes I put unnecessary parentheses to enhance clarity, and, looking at it myself I don't know whether it enhances it or not. Would you like parentheses to the left of the quote and directly to the right of the DD to say that thing in parentheses is my function? I don't know, does it help or hurt? (mixed replies from the audience).

Let me show you some other things, what you should do when you get started. This is what Bob Bernecky said APL2 didn't have; well maybe he's right but with DD I can do this without having to define anything, so APL2 does have it. This is the cartesian product of two lists of things

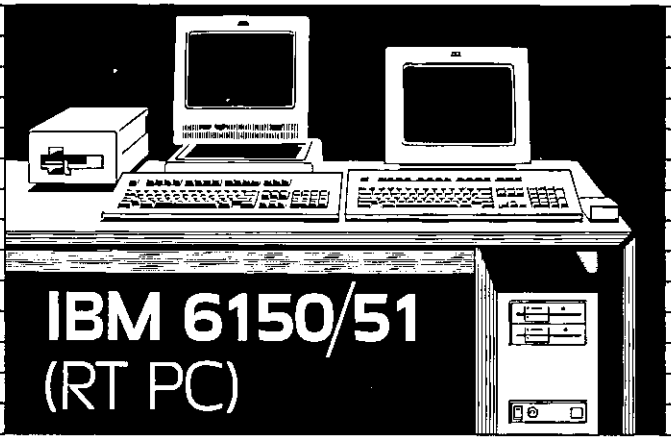
```
    'APL' 'AI' ∘.('⍺ ⍵' DD) '' 1 2 86 (2 3⍴⍳6)
APL      APL 1     APL 2     APL 86     APL   0 1 2
                                          3 4 5
AI       AI 1      AI 2      AI 86      AI    0 1 2
                                          3 4 5
```

and you can see APL being paired with each item on the right in every combination using outer product, so now with APL2 outer product can be used any time you need any sort of cartesian product. Doesn't have to be a scalar function, it can be any function, and this says make a 2-item list, take one from the left and one from the right. Now the way you do it without DD is you enclose each of the left and you enclose each of the right and you do jot-dot-comma. Why do you need to enclose it in this case? Because you get a length error trying to catenate things of wildly different shapes. If you have, say, character strings and you want to glue them together in every combination, just simple jot-dot-comma will work. So that's a pretty easy way.

And if it will make it a little bit freer, I'll show you that you get a 2 by 4 array, which you can deduce from the original outer-product shape rules: the shape of the result of an outer product is the shape of the left catenated to the shape of the right. I also find you can do useful work. One of the common phrases in my paper is the first of jot-dot-comma-reduce and one application I use of that is I do an outer-product reduction, I use that to form all the indices of an array using jot-dot-comma-reduce, so this does come up in day-to-day work, I'm not just making pretty slides.

dyalog APL

IS AVAILABLE **NOW** FOR THE HIGH PERFORMANCE



IBM 6150/51 (RT PC)

AND FOR 22 OTHER **UNIX*** SYSTEMS!

The IBM 6150/51 is a new family of high performance 32-bit multi-user micro computers and workstations. It is based on reduced instruction set (RISC) technology and AIX**, IBM's enhanced version of UNIX System V; and is an excellent basis for a Dyalog APL system.

Dyalog APL on the IBM 6150/51 is fast, especially with the floating-point accelerator, and there is practically no limit on the size of APL objects and workspaces.

The entire range of standard IBM screens and printers for the 6150/51 is fully supported, as are most of the common APL/ASCII vdus.

All AIX facilities can be accessed from within a Dyalog APL function and results may be captured in an APL array. Compiled subroutines written in C or Fortran can easily be incorporated into applications, and interfaces between Dyalog APL and several IBM supported software packages such as SQL and the Professional Graphics Series are available.

Dyalog APL adheres to the draft ISO standard for APL and includes many of the features of APL2. Nested arrays, and corresponding extensions to primitive functions and operators, greatly simplify the processing of lists and non-rectangular data structures. Dyalog APL contains a host of useful features including a session manager, full-screen editor, and full-screen data manager.

The IBM 6150/51 with Dyalog APL is available direct from Dyadic. Prices, including Dyalog APL, start from around £15,000.

For further information, please contact ***Peter Donnelly***
Dyadic Systems Ltd., Park House, The High Street, Alton,
Hampshire GU34 1EN, United Kingdom
Telephone: Alton (0420) 87024 (10 lines). Telex 858811

*UNIX is a trademark of AT & T Bell Laboratories. IBM is a registered trademark of International Business Machines Corporation
**AIX is a trademark of International Business Machines Corporation.

ANNOUNCING
THE

COPROCESSOR
FOR THE **IBM PC**

The Dyalog APL coprocessor allows large scale second-generation APL applications to be developed and run on an IBM PC or plug-compatible computer.

The Dyalog APL coprocessor consists of:

- **a 32-bit coprocessor board based on the NS 32000 chip set, hardware floating-point, up to 4Mb of on-board RAM, and 16Mb of virtual memory space.**
- **a complete implementation of UNIX* System V including C, f77, Fortran and Pascal.**
- **a complete implementation of Dyalog APL.**
- **software that integrates the Dyalog APL coprocessor subsystem into the PC-DOS environment.**
- **a Dyalog APL character generator and set of keycap stickers.**

The Dyalog APL coprocessor is easily installed in an IBM PC or plug-compatible system with a standard BIOS and a hard disk. The board simply slides into a single slot in the PC chassis, and the software is installed via an interactive menu.

The Dyalog APL coprocessor provides both DOS and UNIX System V environments without compromise. Data can be shared between the two environments, and a simple keyboard command switches between operating systems. DOS commands and files can be accessed directly from Dyalog APL, and Dyalog APL can be run directly from DOS.

The Dyalog APL coprocessor supports very large APL objects and workspaces, to a maximum of nearly 16Mb; and floating-point performance is up to 3 times faster than an IBM PC AT.

Prices, including a multi-user Dyalog APL licence, start from around £3,000. A complete ready-to-run system with a fast 40 mb disk and tape back-up is also available for under £7,000.

For further details and ordering information, please contact **Peter Donnelly**
Dyadic systems Ltd., Park House, The High Street, Alton,
Hampshire GU34 1EN, United Kingdom
Telephone: Alton (0420) 87024 (10 lines) Telex 858811

Let me show you another application of DD which I do use all over the place to such an extent that I'd like to have some built-in facilities that will do this without my having to carry around DD. I'm not sure how that would work, but I'd also like to do without the quotes, I'm not sure how that would work but there are proposals. In teaching APL Ken (Iverson) suggests doing outer products to see tables.

```
      ¯1 0 1 2 ∘.* 0 1 2 3
1  ¯1  1  ¯1
1   0  0   0
1   1  1   1
1   2  4   8
```

Maybe somebody else asks where do those answers come from. You'd like to trace star. So you'd like to have a variant of star, which shows what's going on as you do it. So I can use the simple recognition method and I say quote-quad DD. Well quote-quad is a sort of funny input – open the keyboard and let me type; and I say all right do the operation and put the answer there, show an assignment, put the left there and the right there and you get this beautiful traced display

```
      ¯1 0 1 2 ∘.(⍞ DD) 0 1 2 3
(⍺*⍵)'←' ⍺ '*' ⍵
1 ← ¯1 * 0   ¯1 ← ¯1 * 1   1 ← ¯1 * 2   ¯1 ← ¯1 * 3
1 ← 0 * 0    0 ← 0 * 1     0 ← 0 * 2    0 ← 0 * 3
1 ← 1 * 0    1 ← 1 * 1     1 ← 1 * 2    1 ← 1 * 3
1 ← 2 * 0    2 ← 2 * 1     4 ← 2 * 2    8 ← 2 * 3
```

Never do my fingers leave my hands. Oh, now I know where they come from

Looking clockwards again – how much time – less than 10 minutes – ah, I see – I thought you were going to say less than 10 nanoseconds. I think two more topics. One of the key things I have to do, If I write a program that's going to be inserted into the interpreter, as a new function, maybe experimentally (we call it PDF – pre-defined function or primitive defined functions) I have to be as severe as a primitive function and fully check the arguments.

```
0=⍴X                    ⍝ Empty
1≥≡X                  ⍝ Simple
X≡⌊X                  ⍝ Integer
X≡⍕X                  ⍝ Simple Character
0∧.=∈⍬⍴0⍴⊂X             ⍝ Numeric
''∧.=∈⍬⍴0⍴⊂X            ⍝ Character
1=⍬⌷⎕EC'⍴⍴X⌈X'          ⍝ Non-Complex
```

So I have a set of idioms for domain checking and of course common ones like this everyone knows (empty), Simple – is the array a simple array? This is integer, assuming that you already know your array is numeric, the floor of all integers is the same array. Here's a simple character one. Then they get massive. X there is an array of any depth rank shape or whatever entirely numeric. Now I found that I could write this over and over again without flaws but it does get a little tricky. Also all character; here's the most tricky one of all – non-complex – meaning X is numeric of any depth rank and shape, but I don't want any complex numbers sneaking in there. What I do is use ⎕ EC, and rank is simply to say look I don't want WS FULL on my ⎕ EC, so rank gives you that single number or maybe it won't work but, maximum is not defined on complex numbers, because which is bigger:

2J93 or negative-15J22? Well, we could have said you take the magnitude from the origin and return the larger one, instead we said we don't know what that means. And so that blows up the maximum, but this is bad because if somebody comes along and says this means this, then this won't work. So that's the quick way to find out non-complex.

I don't like doing this, because now I know what to say; I'd rather say the word EMPTY or the word SIMPLE and so forth. I don't like to do these phrases because I can make a mistype, I can leave out this enclose, and now it works sometimes, I'd rather just say this. So, how do you do this? You can just say it. The nice thing about programming languages is that if you don't like a facility that's there you can abstract it by making a program. This is what I call a 'good' program (I say good in quotes because if you're not making a program that you want to be just like a primitive and reject all its arguments you don't have to be so extreme)

```
  Header
⍝ Abstract
⍝ Prolog
  Argument checks      ⍝ Comments
                       ⍝
  Body                 ⍝
                       ⍝
```

You have an abstract, a one-line abstract; a prologue saying don't do this with it or that, or maybe an example; then you do argument checks, than you have code that actually does it. So let me give you an example of why you might want to do this I've purposely chosen a stupid function to do. Now COUNT means in origin 1 iota-N; it's so simple even in origin zero you don't want a subfunction. I'm showing this to emphasize that argument checking helps. So I test it like a good programmer – COUNT 3 and I test it again, how about something bad and you get an answer

```
  [0]  Z←COUNT N
  [1] ⍝ First N counting numbers
  [2]  Z←+\Nρ1

      COUNT 3
1 2 3
      COUNT 2 3
1 2 3
1 2 3
```

Now that may not be desirable; we want to say LENGTH ERROR, or DOMAIN ERROR or some such message. So how do I do it? OK I'm a thorough person, I'll go through and put the checks in.

```
[0]  Z←COUNT N
[1] ⍝ First N counting numbers
[2]  ⎕ES(~1=⌈/ρN)/5 3
[3]  ⎕ES(~1≥≡N)/5 4
[4]  ⎕ES(~0=↑0ρN)/5 4
[5]  ⎕ES(~1≡⌊N)/5 4
[6]  ⎕ES(~N≥0)/5 4
[7]  Z←+\Nρ1
```

```
      COUNT 3
1 2 3
      COUNT 2 3
LENGTH ERROR
      COUNT 2 3
      ^
```

Do you have any uneasy feeling at all about this function? (laughter) Yeah, even if you don't care about it very much, I look at this and say Where's the code that actually does it? There's several nice things that I can say about this, the error-checking part of it is in the beginning all with ⎕ ESs, so I could work out a workspace with all the checks in and then run a processing function that takes them out once the thing works. So we have in APL a strongly-typed language if we want a strongly-typed language. And you also notice that these phrases aren't horrible error-trapping phrases, but you could make a mistake, you could do this and put 1 less-than depth and get wrong; I do try to do this as NOT, this means that if it is NOT a single then blow up with a length error. If it's NOT simple blow up and so forth. As for efficiency, I don't care, NOT is very very fast, especially NOT on 1-bit scalars. In fact, I was showing off APL PC 2.0, with a 30,000-element bit vector...how many more minutes – I won't tell that story.

So this works just like the primitive would work with different argument. How can we do a little better? Well, look, I can actually apply IOTA, I can apply a related function that has the same domain, and if it blows up for any reason I can use the error we're going to code. So the style is this, now it's an admittedly bad example because COUNT is trivial, it's using IOTA to check essentially IOTA, but I've used this in good cases. I proposed a new function whose argument treatment and forms is just like compression arguments and forms. So how do I test that they're OK? I try compression on the arguments, if they blow up I say No good. They're easy to do, actually they're quite fast, there it is. I could make one more step, of course this was not my point, but I can do it with IOTA, and I want it to accept any single matrix or scalar or vector, so I can do it like that, and now at this point we spit back the errors, even the one – resource failure, we spit back just like a primitive that fails to work.

```
[0]  Z←COUNT N
[1] ⍝ First N counting numbers
[2]  '⎕ES ⎕ET' ⎕EA '→0⍴⍳,N'
[3]  Z←+\(,N)⍴1
```

Now there is a better way, and a fast better way.

```
    [0]  Z←COUNT N;⎕IO
    [1] ⍝ First N counting numbers
    [2]  ⎕IO←1
    [3]  '⎕ES ⎕ET' ⎕EA 'Z←⍳,N'

      COUNT 3
1 2 3
      COUNT 1 1 1⍴4
1 2 3 4
      COUNT 'OF MONTE CRISTO'
LENGTH ERROR
      COUNT 'OF MONTE CRISTO'
      ^
      COUNT 2*35
WS FULL
      COUNT 2*35
      ^
```

Finally the function COUNT is perfect

1. Correct output given proper input.
2. Rejects all improper input.
3. Reports resource failures.
4. Easy to understand. *
5. Efficient. **

(* Some may argue about this)
(** Usually overemphasized)

I've made the two last points 'Easy to understand' that's heavily subjective, I tried to make it easy to understand. Efficient? – well, probably overemphasized. Many times I find my watch doesn't go down to nanoseconds so I'm not sure if it's inefficient or not. There is another way to do it. I could write helper functions – I'll show you the helper functions later. Now you do this, and you say What the heck is this PL/1 doing in my APL code.

```
    [0]  Z←COUNT N
    [1] ⍝ First N counting numbers
    [2]  DECLARE NONEG INTEGER SINGLE SIMPLE N
    [3]  Z←+\(,N)⍴1

      COUNT 3
1 2 3

      COUNT 2 3
DOMAIN ERROR
COUNT[2]  DECLARE NONEG INTEGER SINGLE SIMPLE N
                        ^       ^

      COUNT ¯2
DOMAIN ERROR
COUNT[2]  DECLARE NONEG INTEGER SINGLE SIMPLE N
          ^       ^

      COUNT 1 (2 3)
DOMAIN ERROR
COUNT[2]  DECLARE NONEG INTEGER SINGLE SIMPLE N
                                ^       ^
```

except for the last name which is the argument name all of these are monadic functions; except for the DECLARE function, all the rest take their argument and immediately return it, but before they do that they check to make sure that it's satisfied. If it's not satisfied they report an error and what happens here is that this is not a single number so you notice that the right-hand caret points to the word that is offended by that argument. Now if everything is clear the N tumbles through from function to function, of course DECLARE is a monadic function that says Oh, we got there, it does nothing. So you can write a very wordlike form.

The other nice thing is that if I put the word NONCOMPLEX in and later on we change maximum so that it works with complex, then I can change NONCOMPLEX without changing this. The other nice thing is that I get everything debugged here, and I don't want to carry around these helper functions with me, then I just go through and put a lamp here and replace that DECLARE and now there's a nice comment in ENGLISH. I also thought this would be wonderful for the compiler people, because if you wrote your programs like this they could specifically recognize a line starting with DECLARE and now they know what the arguments are, which is one of the hard things to do for a compiler.

Let me sum up. Finally one more step in this. You notice that we showed the inside of the code. Finally after we're sure that everything works we want to say like Peter suggested, Don't show me that, report it like a primitive, We can finally do this with ⎕FX ⎕CR

```
      0 1 0 0 ⎕FX ⎕CR 'COUNT'
COUNT

      COUNT 3
1 2 3

      COUNT 2 3
LENGTH ERROR
      COUNT 2 3
      ^

      COUNT '?'
DOMAIN ERROR
      COUNT '?'
      ^
```

Now we'll put up at the end after I'm done the functions that do that. I think that's far more readable and I am an APL bigot and I prefer to say it that way than say it another way. You don't have to make any extensions to the language, so I'll give you these functions, they're one-liners after the comment.

Let me leave you with a quick whizbang if I may. This is one that we noticed I think, well Roy Sykes gets the credit for this perhaps. This says Is A and B the same to 4 significant digits?

```
⁻4 ≡ (▼⍤⁻) A B
```

and it works like this: I can rewrite it like that

```
≡/⁻4 ▼⍤ A B
```

which says this

```
≡/(¯4↑A)(¯4↑B)
```

and now you can see, this

```
⊂(¯4↑A)≡(¯4↑B)
```

and match always gives back a simple scalar so you don't need the enclose

```
(¯4↑A)≡(¯4↑B)
```

and obviously this matches the arrays: are they all to 4 digits? But if I don't want to check that, I want to say How many digits do they match to? and so now what I need is to say "for D, I want to know D."

```
(⍳D)≡⌿(↑¨) A B
```

I could do this

```
(⍳1)≡⌿(↑¨) A B
```

I could do this

```
(⍳2)≡⌿(↑¨) A B
```

I could do this

```
(⍳3)≡⌿(↑¨) A B
```

until they finally match, but I have a list of numbers of digits, like 1, 2, 3 and so forth, so I can do it with a whizbang, I don't call it an idiom it's not common enough, I use the All Right, and match-dot-format-each is the function that I do All Right with and now what do I do? To find out how many digits? Just sum them up

```
      (⍳+\18⍴1)≡⌿(↑¨)¨ ⊂A B
1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0

      +/(⍳+\18⍴1)≡⌿(↑¨)¨ ⊂A B
5
```

so let me leave you with something that. . .sometimes it hurts me when I try to think . . . but here is an APL2 whizbang.

```
+/∧\(⍳+\99⍴1)≡⌿(↑¨)¨ ⊂A B
```

I figure 99 digits: all systems would poop out before 99 digits. So now, the and-scan there is to say take the format and if you get a one after you get the first zero then don't consider that. I showed this at New York City SIGAPL meeting and somebody said "Oh my goodness", I felt that person was representing the computer this was run on. (laughter) I will . . . to you, this does turn on the error-conditioning. But you have to realize this, I don't mind how much CPU time is used up, because if you don't use the CPU time now, it's gone for ever. (Laughter)

O.K. (Applause)

BACK NUMBERS OF VECTOR

Back numbers of VECTOR are available from the BCS. If you don't have them all, now is the time to complete your collection. Apart from the technical contents, every issue includes book and product reviews, letters, news and a competition. Send in your order before they run out. These will one day be unobtainable collectors' items, like the early issues of Quote Quad.

The prices inclusive of postage and packing are as follows:

	Prices in Pounds Sterling		
	UK	Surface (inc. Europe)	Airmail (outside Europe)
Single issues	3	3.75	5.75
Volume 1	10	14.00	22.00
Volume 2	10	14.00	22.00

Please send sterling cheques or money orders (payable to The British APL Association) to the Treasurer:

Mel Chapman, 12 Garden Street, Stafford ST17 4BT.

Don't forget to include your name and address and to be clear which VECTORs you want.

Introduction to General Articles

This issue of VECTOR contains two articles of a non-technical nature.

The first article continues our extracts from Anthony Camacho's series *Steps to a better BASIC* first published in Datalink. This series explains some of the concepts behind, and advantages of, APL to an audience brought up on the staple diet of BASIC available on most microcomputers. Anthony's article is reprinted by kind permission of *Datalink* magazine.

Many of you will remember Graeme Robertson for his erudite and entertaining offerings at a number of BAA meetings. Graeme recently left I P Sharp Associates to work for a PhD in particle physics at Durham University, but before he left he held a seminar, intriguingly titled 'APL3'. (Maybe soon we'll reach APL86). Graham Parkhouse attended the seminar, and as well as telling us about it, philosophises on the future directions of APL.

Steps To A Better BASIC

by Anthony Camacho

Everything AND the kitchen sink
or how to take it with you when you go

When you go caravanning, instead of taking tents out of the boot of the car and erecting them in the pouring rain, it seems like luxury. It's not the same, of course, as doing things the hard way, but it does make life easier.

The COBOL or BASIC way of taking things with you is to put them into constants or DATA so you unpack the luggage freshly on every run. This is fine for a data processing application, but less than ideal when it comes to programming.

That calls for easy ways to handle parts of the program such as subroutines. A COBOL library is easy. Most BASIC methods are not. COBOL writers can copy from the library at compile time, so there is no problem with clashes of line number. BASIC programmers have to write their standard subroutines with high line numbers so that they can be merged with any main program.

Programmers in most BASICs do not have the facility to call subroutines by name so they get to know the line numbers of the main routines and of course they get used to a particular line number doing each of the common things they want to do. For example GOSUB 9873 may display M$ centred on line 21 flashing with three beeps. The main program can't be tested without the subroutines; after they are merged it can't be renumbered because that would move all the subroutine addresses.

Where BASIC does have an advantage over COBOL is in the programmers' toolkits which provide all kinds of useful debugging aids. In microcomputers this may be held in ROM so that it is always available. This brings such joys as the ability to display the list of all variables and their current values or to trace a variable's changes of value.

In APL on the other hand it is easy to take things with you. Indeed that is the default. All APL work is done in a notional "workspace" which is a block of real or virtual memory. The workspace holds your bits of program (which are all in the form of procedures or "functions" as APL calls them). It also holds all the variables that have been assigned a value. At any time you can get a sorted list of variables with the command)VARS or a list of functions with)FNS. The workspace also contains the stack, which records all the return addresses for functions in progress and such "global variables" as the print precision, the print width, the comparison tolerance (yes APL finds a million millionths equal to 1), the seed for pseudo-random numbers and so on. If you interrupt the execution of an APL program and)SAVE the workspace, everything is stored, and branching to the line counter after reloading it will carry on exactly as if there had been no interrupt at all.

You can copy functions and variables into your workspace from another workspace if you need, so any useful tools (such as a full-screen editor or a format controlled listing function) can be copied when needed and either kept in the workspace or erased from it before it is saved.

This approach removes most of the pain of holding the latest run date, master file identity and any current parameters from one run to the next – they are simply stored in the production version of the workspace which automatically can save a new version of itself each time it runs. There is no need for those troublesome bits of program to store such details in one of the master files in a place where they will be accessible from the beginning of the next run.

APL provides filing systems – and often it isn't practical to hold all the data in the workspace. If the records are extremely large it may even be necessary to read them in and process them one at a time. But for most purposes the workspace will suffice. Even on microcomputers quite respectable amounts of data can be held without bothering with files.

My BBC microcomputer a version of APL running and the workspace limit is 400K – about the size of a respectable book. On the IBM PC workspaces of up to half a megabyte are common, and with the 68000 they can be as large as you need, up to the address limit of 16 megabytes.

In short the workspace is ideal for programmers. It saves trouble, simplifies manipulation of programs and data, and assures consistency between runs. Why doesn't any other language have one?

Time to think about the future directions of APL

by Graham Parkhouse

Not long after returning from APL86 in Manchester this July, I received an invitation to a one-day seminar called *APL3 – a seminar on the future directions of APL*. Now I am a comparative new-boy to this APL terminology; APL86 was the first annual APL conference I had been to. From the conference I had heard so much about APL2 and second generation APLs that I quickly discerned the implication of the APL3 in the title to the seminar. Being an engineer, I am a practical man who might be more attracted to APL3 were he persuaded that the number attached to the APL referred to the language's horsepower. But, being theoretically inclined also, I am fascinated by APL as a phenomenon and needed no encouragement to join a discussion on its merits and its future directions.

I want to tell you a little about the seminar and to share some of my thoughts about APL development, but before doing so let us consider some practical issues surrounding second generation APLs. By "second generation APL" we mean an APL implementation that includes nested arrays, i.e. arrays of arrays. With the nested arrays come many new primitive functions and operators. Additional features of most second generation APLs are the facility for using primitive operators with user-defined functions and the facility for user-defined operators. There is no generally accepted standard for second generation APLs, and all those currently being used do differ significantly from each other. This divergence in the development of APL is weighted by IBM's presence; due to their influence, APL2, which is their particular dialect, is likely to become the industry standard. What is more, "APL2" is already in popular use as the generic name for second generation APLs, just as "APL1" is being used as the generic name for first generation APLs. So the name "APL2" is likely to continue to have dual meaning in the same way as well-known brand names like "Biro" and "Hoover".

I suspect that the title of this seminar I am going to tell you about was designed to get our attention OFF APL2, being organised as it was by I.P.Sharp Associates, a company who have done so much to promote advanced computer systems and who, in the process, have promoted APL. Their own Sharp APL was the first commercially available second generation APL. But the seminar was not about Sharp APL; Graeme Robertson, of I.P.Sharp's education section, presented a survey of APL principles which embraced more aspects of APL than I had ever thought of. Graeme obviously appreciates structure, because he had cleverly organised the whole day's contents into a special ternary tree structure, subdivided to five levels, each subdivision containing three items. This resulted in 3*5 pieces of information to be delivered in three hours at an average pace of one piece every 45 seconds. You will be glad to know that he was very flexible in his delivery, taking questions and diversions in his stride, but he had to triple his rate after lunch. (Note the factor of three.) But he managed it so well that we were able to enjoy the virtuosity of his delivery as well as the quality of the content.

What did he say? He talked about parts of speech, cells and frames, parsing rules; he introduced me to the concepts of potency and weights of operators; we discussed direct definition of functions and covered some of the new primitive functions described in Iverson's *Dictionary of APL*. Graham went on to demonstrate how APL has been used to express all the well known scientific theories most concisely, and tried to demonstrate that APL was relevant to many of the popular mathematical techniques such as predicate calculus.

It was at this stage that I realised that we were not going to be introduced to the nub of APL3 (in the same way as nested arrays are the nub of APL2). Instead Graham gave us an illustrated glimpse into topics of current research interest: data representations, tensor analysis and functional analysis, finishing with an introduction to fractals, a new branch of mathematics concerned with hierarchical patterns. He emphasised the importance of computer graphics, and suggested that future usage of APL would be more graphically orientated. I was challenged by the need for mathematical tools that will help us understand and manipulate the data supporting these concepts, tools as elegant as the concepts themselves. Can APL meet this need?

Who can be sure? What is certain is that practitioners of APL are breaking new ground. APL is unique in being the first, the most highly developed, and the most used executable analytical notation. Being a symbolic analytical notation it matches our thought processes which are themselves symbolic, and being executable it gives us the benefit of experimentation; in other words a mighty tool for intellectual explorers.

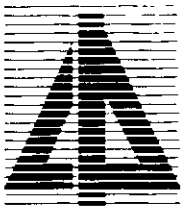
I cannot forget Phil Smith's presentation to APL86 called *A Programming Language for Thoughts and Dreams*. Those who possess the Tutorial Volume of the Proceedings can enjoy experimenting with the random dot pattern illustrated in his paper. The illustration is a collection of black, red and blue dots randomly distributed over a rectangular area. Looked at normally there is no evidence of any pattern whatsoever within the picture; the rectangular boundary is the only shape, and all dots seem evenly distributed. But looked at through the coloured glasses provided, rows of steps are apparent, with the lowest steps along the top and the bottom row, rising to higher steps towards the middle row, which itself is the highest step. Without the glasses there is no evidence of these steps. But I have not mentioned Phil's purpose behind the demonstration which has so impressed me. This was to demonstrate the working of the right-hand side of our brains. When you first look at the picture through the glasses, nothing happens! You have to study it for several minutes before the steps start to take off into three dimensions. During these several minutes your brain is matching the dots seen by one eye with those seen by the other (because with the glasses each eye is seeing a different pattern) and decides that the patterns would be identical if they were not on a flat piece of paper but on rows of steps. Do not worry if you do not understand my explanation; the point is that our brains perform a very sophisticated calculation without us being conscious of what it is. We do not know what our brains are doing, but they do it, and it takes them time.

How often have you stopped grappling with a problem on your mind and turned to other things, when suddenly you present yourself with a solution to your original problem? Inspiration? Yes, but probably only after the right side of your brain had done a lot more work on the original problem while you were consciously thinking about the next one. It does seem that the right side of the brain is a parallel subconscious processor with the left, more conscious side. I am not suggesting that we all have split personalities; both sides have been designed to work harmoniously together as a team, probably with the left side dominant. Following Phil's recommendation I got hold of Betty Edwards' amazing book *Drawing on the Right Side of the Brain* (see reference) – a book that has not only taught me about how to draw, but which has demonstrated to me the potential struggle between the two halves of the brain. She explains why most of us are so bad at drawing; it is because of our impatience. In our impatience the left sides of our brains keep guiding our hands to draw stereotyped symbols for each familiar object, such as the sun with its childish rays; when we inhibit the left side, and she explains how to do this, then the right side is given a chance to display its natural artistic ability. With practice we can encourage the right side to do what it is designed to do, and stop the left side from being over-dominant.

What has this got to do with APL? A great deal if, as I suspect, our understanding of mathematics is shared between the two sides of our brains. Then we should expect to need time to assimilate ideas, time to gain sufficiently deep understanding to be able to recognise the future directions of APL. Through their pioneering, Ken Iverson and his collaborators have given us a most valuable means of expression with which we may experiment and learn. APL1 has modified the way many of us think. Developments in APL1 will lead us on further, but I believe that what we need most is to use what we have, at the same time gaining deeper insight into the problems facing us. APL development is a time-dependent process; dependent on the time it takes our subconscious minds to assimilate new ideas, which should not be hurried.

Reference:

Edwards, B. *Drawing on the right side of the Brain* J.P. Tarcher Inc., Los Angeles.



COCKING & DRURY LTD.

THE APL PROFESSIONALS

16 BERKELEY STREET · LONDON · W1X 5AE

TELEX: 23152 MONREF G
TELEPHONE: (01) 493 6172

COURSES

APL Fundamentals Jan 20-22	3 days	£375.00
APL*PLUS PC Intermediate Jan 26-29	4 days	£525.00
APL Fundamentals Feb 10-12	3 days	£375.00
APL System Design Feb 16-19	4 days	£595.00
APL Fundamentals Feb 24-26	3 days	£375.00
APL Fundamentals Mar 10-12	3 days	£375.00
APL Fundamentals Mar 24-26	3 days	£375.00

Discounts are available to companies making more than one booking at a time:

Number of bookings	Total Discount
2	£50 (ie £25 per course)
3	£90 (ie £30 per course)
4	£140 (ie £35 per course)

If you require courses on APL2 or other specialised topics we will be happy to provide them on your premises. In-house courses are also likely to be cost effective if you have 4 or more people to train at any one time.

All prices exclude VAT.

For further information, please ring Liz Swann on 01-493 6172.

TECHNICAL SECTION

This section of VECTOR is aimed principally at those of our readers who already know APL. It will contain items to interest people with differing degrees of fluency in APL.

Technical Editorial: Interpreters for Debuggers

by David Ziemann

Those of us who use APL to develop applications already know that the language offers remarkable reductions in implementation time when compared with other languages. What effect does using APL have on the other phases of system development, and in particular, how helpful is APL during debugging? Let us remind ourselves of the various stages in the life of a application. Broadly speaking the application development process can be broken down into the following nine tasks:

- Determine requirements
- Specification
- Design
- Implementation
- Testing
- Debugging
- Documentation
- Maintenance
- Modification

No task is truly independent and a definitive order should not be implied – documentation for example, may well be developed in parallel with other phases.

Direct use of interpreter facilities is only strictly necessary during the implementation, debugging, maintenance and modification phases, although APL may also be used to help in other ways. (An APL prototype for example, can be considered as a specification for a final system). Maintenance and modification can be viewed as similar activities to implementation, where APL is used to build, correct and extend programs. When APL programs go wrong, the user still remains in the APL environment, and so the interpreter is necessarily used during debugging. The debugging process, however, does not usually require the construction of programs, but more often depends on the use of facilities available in immediate execution mode. The ability to view the SI stack, to look at the names and values of variables and even to change their values are all debugging tools which are 'naturally' available. Trace and Stop facilities are provided in most APLs as system functions or via the T-delta and S-delta syntax. Trace and Stop seem however to be the only tools available explicitly for debugging, and Stop has even found other uses outide this area.

Do these typical debugging facilities match up to the typical problems that one experiences during debugging? The sort of questions that we want to answer are; "A spurious 1 appears on the screen while my system is running. Where was it produced?", or "Where on Earth did the variable <flag> get set to 17?", or "How did this simple expression produce this strange result?", or "My system accidentally leaves a variable <I> as global. In which function did I forget to localise it?". Although the answers to such questions may usually be determined by a combination of esoteric programming combined with trial and error, the interpreter does not make it easy for us.

APL interpreters do not appear to be improving in this area, in fact there is evidence to the contrary. Well over ten years ago Xerox's Sigma APL included the system commands)OBSERVE and)CATCH.)OBSERVE caused the subsequently executed APL expression to be 'observed', ie for every intermediate result to be displayed under a caret line indicating the progress at each stage.)CATCH allowed the programmer to trap the assignment of a variable. The command)CATCH X VIA FOO caused the function <FOO> to be executed whenever the variable <X> was assigned a value. Very useful indeed, but I've never seen it anywhere else since. It might be possible to provide this facility in systems that support exception handling by considering assignment as a type of exception.

Other desirable debugging facilities include the validation of newly defined functions for simple syntactic errors, global references and assignments and even clashes between local names such as labels with other names. The ability to 'travel' through the SI stack environments to examine their local contexts would also be valuable.

One objection to the provision of these, and other, debugging tools is that they result in unacceptable performance penalties during production use. If this is true, then solutions must be found which enable debugging aids to be delivered to the programmer. One approach, particularly in the PC environment, might be to provide two interpreters – one which includes a whole family of debugging facilties, and one without the features, the production interpreter. Which interpreter is used could be decided by an APL invocation option, or perhaps more flexibly by running an 'interpreter generation program' which would produce one of the two interpreters as its output.

It is clear that implementers have concentrated their efforts in encouraging the programmer to reduce the cost of the implementation phase of a project by providing high-performance tools such as nested arrays, full-screen I/O facilities and exception handling, among others. Now is the time for them to similarly enhance our debugging tools so that savings can also be made in this area.

Technical Correspondence

Watch Your Step

From Neil Mitchison 17 July 1986

Dear Editor,

I enclose a submission for the 'Watch Your Step' competition. I am afraid that at present I have no facilities for printing APL, so I have had to write out the function in longhand. However I hope that will not give Dave Ziemann too many problems. I have tested the function on Version 2.2 of Siemens APL, running on a Siemens mainframe (which is in fact a 360 lookalike).

I think the code is fairly explanatory; while the number of lines could be reduced, it would be at the expense of clarity, in my opinion. The same goes for reducing the number of local variables.

Obviously the problem would be trivial in a nested-arrays version.

Yours sincerely,

Neil Mitchison
Bremlaan 55
1900 Overijse
Belgium

(Editor: Thanks for your letter Neil, and thanks also for colouring the zeroes pink in your entry – it would otherwise have been quite difficult to distinguish them from the letter Os in your local variable names! The result of the Watch Your Step competition appears later in this issue.)

APL2 bugs

From David Piper 10 October 1986

Dear Editor,

Some APL2 bugs to add to your (no doubt extensive) collection. All the below encountered in APL2 1.2.

1. EDITOR 2

   Inserting lines after line 7. Lines N.1, N.3, N.5, N.6 and N.8 are fine . . . but try N.2, N.4, N.7 and N.9, and the result is a definition error. The more digits after the decimal point, the more confusing the rules get. See listing 1 for an illustration.

2. Assember Function DAN

   Bug or design feature? Adjacent delimiters do not result in items of zero length – the items are omitted (listing 2).

3. APL WSID

Command of the APL2/TSO interface returns a three row array, first row contains the WSID, the second the time stamp of the last save and the third the userid of the perpetrator of the said)SAVE. If an object is copied from another WS, the time stamp and userid are changed to that of the WS from which the object was copied (listing 3).

4. APL QUIET

Command of the APL2/TSO interface is documented as stopping output to the terminal until a prompt for input is issued or immediate execution mode is re-entered. APL errors and STOP messages turn the quiet mode off (listing 4).

At the current rate of discovery, more next time!

Yours sincerely,

D.B. Piper
Rainham, Kent.

Listing 1. Editor 2 Definition Error

```
[⍵]∇ TEST.2  ⍴: 8 14
[0]   TEST
[1]   SOME TEST TEXT
[2]   SOME TEST TEXT
[3]   SOME TEST TEXT
[4]   SOME TEST TEXT
[5]   SOME TEST TEXT
[6]   SOME TEST TEXT
[7]   SOME TEST TEXT
[8]   SOME TEST TEXT
[7.1] THIS LINE WILL BE OK
DEFN ERROR
      [7.2] THIS WILL GIVE DEFN ERROR
```

Listing 2. Assembler function DAN

```
      )CLEAR
CLEAR WS
      3 11 ⎕NA 'DAN'
1
      )COPY 1 DISPLAY DISPLAYG
SAVED 6.06.1986  8.59.09 (GMT)
      ⍴⍨ '/' DAN 'ITEM1/ITEM2//ITEM?'
 5  5  5
      ⍝ THE NULL ITEM IS EXCLUDED
      3 11 ⎕NA 'CAN'
1
      ⍴⍨ ('/'≠'ITEM1/ITEM2//ITEM?') CAN 'ITEM1/ITEM2//ITEM?'
 5  5  0  5
      ⍝ THE NULL ITEM IS INCLUDED
```

Listing 3. APL WSID being corrupted

```
      )LOAD TESTING
SAVED 28.07.1986  9.25.57 (GMT)
      )HOST APL WSID
TESTING
28.07.1986 09.25.57
V500003
TSO(0)
      )COPY ROB1
SAVED 22.09.1986 10.26.39 (GMT)
      )HOST APL WSID
TESTING
22.09.1986 10.26.39
V500015
TSO(0)
```

```
Listing 4. APL QUIET turned off by stop message

      ∇TEST[□]∇
    ∇
[0]   TEST
[1]   □←'BEFORE QUIET'
[2]   RC←⎕TSO 'APL QUIET'
[3]   RC←RC,⎕STACK'''IN STACK''' ')WSID' '→□LC'
[4]   □←'AFTER STACK CMD, BEFORE STACK RUN'
[5]   S⎕TEST←10
[6]  10:S⎕TEST←↓0
[7]   □←'AFTER STACK RUN'
[8]  ⍝ The only message generated by this function should be:
[9]  ⍝ BEFORE QUIET          (from line 1).
    ∇ 2.10.1986 12.10.58 (GMT)
      TEST                  ⍝ NOTE: Comments added after execution.
BEFORE QUIET                ⍝ + This message is OK
                            ⍝ + Where does this line come from?
TEST[6]                     ⍝ + Quiet has been turned off
IN STACK                    ⍝ + Stack is run
IS CLEAR WS
AFTER STACK RUN             ⍝ + Function restarts
      RC
 0  0  0  0
```

And some more!

From Colin Jackson 10 October 1986

Dear Sir,

I have noticed the following odd behaviour of APL2 release 2, running under VM:

```
(A B) ← 1 2
```

assigns 1 to A and 2 to B as expected. However,

```
(A (B C) D) ← 1 (2 3) 4
```

sets A to 1, B to 2 3, C to 4, and leaves D undefined. Maybe a SYNTAX ERROR would be better!

Yours sincerely,

Colin Jackson
Cocking and Drury Ltd
16 Berkeley Street
London W1X 5AE

(Editor: VECTOR will publish details of any other APL bugs that our readers discover, so do let us know of any nasties you find. Surely APL2 can't be the only APL system with bugs in it?)

Competition Result – Watch Your Step

by David Ziemann

This time the challenge was to write a function STEP that produces a matrix of 'step' vectors from its argument, a three-column matrix of start, stop, and step values. For example:

```
      □←M1←5 3⍴10 27 5,80 100 20,113 100 3,¯8 ¯3 2,¯1 ¯3 1
 10  27   5
 80 100  20
113 100   3
 ¯8  ¯3   2
 ¯1  ¯3   1
      0 STEP M1
 10  15  20  25   0
 80 100   0   0   0
113 110 107 104 101
 ¯8  ¯6  ¯4   0   0
 ¯1  ¯2  ¯3   0   0
```

The left argument is a pad value for use in cases where a step vector contains zero, as in:

```
      □←M2←3 3⍴8 ¯3 1,¯6 6 2,¯2 2 1
 8 ¯3  1
¯6  6  2
¯2  2  1
      99 STEP M2
 8  7  6  5  4  3  2  1  0 ¯1 ¯2 ¯3
¯6 ¯4 ¯2  0  2  4  6 99 99 99 99 99
¯2 ¯1  0  1  2 99 99 99 99 99 99 99
```

The competition attracted fourteen entries from Australia, Belgium, Denmark, West Germany and of course the UK. (What happened to our US and Canadian readers?). The entrants used eight different APLs to code their solutions, most of them on PCs and micros; VS APL, APL2, Sharp APL, APL*PLUS PC, IBM PC APL, APL.68000, MIPS APL and Siemens APL.

Boiling the entries down proved fairly difficult, mainly because they all worked. At least, they all appeared to work for the above arguments. Closer inspection revealed that one entry did not return an explicit result, and that another went into an infinite loop if the first element of the right argument was a one!

As usual, dependence on the external index origin was tested for, and three entries were set aside because they failed when a global origin of zero was used. The next test tried each entry with a fractional right argument, as in:

```
      M1*2
  5   13.5  2.5
 40   50    10
 56.5 50     1.5
 ¯4   ¯1.5  1
¯0.5 ¯1.5  0.5
      0 STEP M1*2
  5    7.5 10   12.5  0
 40   50    0    0    0
 56.5 55   53.5 52   50.5
 ¯4   ¯3   ¯2    0    0
¯0.5 ¯1   ¯1.5  0    0
```

All the functions bar one behaved appropriately, producincg the result shown above. A floating point left argument produced the expected result in all cases, and so the search for more exacting tests was on.

What would happen if the entries were tried with a vector left argument, rather than a scalar? All entries performed as expected with a one-element pad value, but they split into three camps when a longer vector was used. The first group reported an APL error, the second group ignored the extra elements, using only the first element in the left argument as the pad value, and the third group used up the pad vector elements in a cyclic fashion. It is hard to see meaning in the cyclic use of a vector of pad numbers, and no such function documented this behaviour, and so these entries were rejected. Strictly speaking, the entries that ignore all but the first element are also not quite right – it is actually misleading and potentially dangerous for a function to accept argument values that it does not use. For example, an extended version of STEP that uses the second element of the left argument for a new purpose might now blow up when used as a replacement for the original function. The functions that caused a LENGTH or RANK error report therefore displayed the correct behaviour, and passed this test.

The next test checked to see if each function gave the correct result when a one row matrix of ones was passed in. The right answer is of course:

```
      0 STEP 1 3⍴1
1
```

Surprisingly perhaps, four more entries bit the dust on this one, producing two columns rather than one in the result matrix.

The next test examined the result when the right argument is an empty matrix, with zero rows and three columns. Before reading on, what do you think the shape of the result should be? This one produced no less than four different shapes of empty result and also a few errors. The error-producing functions were eliminated because an empty result is certainly to be expected in this case. Furthermore, we should expect the result to have as many rows as the argument matrix and this criterion eliminated the entry that gave an empty vector as its result.

Of the remaining empty matrices, some had zero columns, some one column and one even had two columns in the result! A zero by zero empty matrix was deemed correct because this result is consistent with the idea that the number of columns in the result should be equal to the length of the longest step vector in the result:

```
      ⍴0 STEP 0 3⍴0
0 0
```

This left us with only two entries. Here is Neil Mitchison's function which has the twin merits of meaningful local variable names and clear APL code. Notice that the argument matrix MX is never indexed in Neil's solution, and that origin-independent code is produced by just one reference to ⎕IO.

```
    ∇ R←FL STEPΔNM MX;GAP;INC;IOTA;LENGTHS;RHO
[1]   ⍝ Produces matrix of step vectors, filled with FL
[2]    GAP←-/ 0 ¯1 ↓MX
[3]    INC←, 0 2 ↓MX
[4]    IOTA←-⎕IO-⍳⌈/,0,LENGTHS←1+⌊|GAP÷INC
[5]    RHO←⍴R←(INC××-GAP)∘.×IOTA
[6]    R←,R÷⍨(⊖RHO)⍴ 0 ¯2 ↓MX
[7]    R[(,LENGTHS∘.≤IOTA)/⍳⍴R]←FL
[8]    R←RHO⍴R
    ∇
```

The other successful entry was submitted by Morten Kromberg, who gave us this function:

```
    ∇ R←FILL STEPΔMK CTL;⎕IO;STEP;START;END;DIFF;N;MAX;MASK
[1]   ⍝(0=⎕NC 'FILL')/'FILL←0'
[2]   ⎕IO←0
[3]   START←CTL[;0]
[4]   END←CTL[;1]
[5]   STEP←CTL[;2]
[6]   ⍝
[7]   MAX←0⌈⌈/N←1+⌊(|DIFF←END-START)÷STEP
[8]   MASK←N∘.>⍳MAX
[9]   ⍝
[10]  R←(START∘.+MAX⍴0)+(STEP××DIFF)∘.×⍳MAX
[11]  R←(R×MASK)+FILL×~MASK
    ∇
```

Morten's function has the additional feature of being able to handle the elision of the left argument when run on an APL system which supports ambi-valent functions.

It seemed clear that the winners had been found, and that their function were complete. I was ready to put the results to bed when another simple test occurred to me. The one row matrix of ones had already been tried, but what about a one row matrix of zeroes? Again, another reasonable function argument. To my horror (and depression because I thought the work was over) twelve out of the fourteen entries failed! Neil, Morten and ten others all yielded the following incorrect result:

```
      99 STEPΔNM 1 3⍴0
0 0
      99 STEPΔMK 1 3⍴0
0 0
```

Of the two who passed this test, one had already failed two other tests, and so this elevated R H Currie's solution, which had produced a one-column empty matrix in response to the empty argument. R H Currie's correct answer to the test, and the well-commented code follow:

```
      99 STEPΔRC 1 3⍴0
0

    ∇ R←L STEPΔRC M;X;Y;⎕IO
[1]   ⍝L is trailing pad number. M[;1 2] are start- and end-points
[2]   ⍝M[;3] are steps
[3]   ⍝Return matrix of range vectors
[4]   ⍝Assume L is numeric scalar; M numeric n×3 matrix
[5]   ⍝Don't mess avout with index origin - set it to 1
[6]    ⎕IO←1
[7]   ⍝Take absolute value of steps: replace 0 with 1
[8]    M[;3]←Y←0=Y←|M[;3]
[9]   ⍝X is no. within range in each row
[10]   X←1⌈(1⌈⍴X)⍴X←1+⌊|(-/M[; 1 2])÷M[;3]
[11]  ⍝Y is a Boolean matrix of required shape: 1=within range
```

```
[12]   Y←X∘.≥⍳⌈/X
[13]  ⍝Generate R as though all ranges are same length
[14]   R←M[;(⌈/X)ρ1]+(M[;3]××-/M[; 2 1])∘.×0,⍳¯1+⌈/X
[15]  ⍝Replace out-of-range elements by L
[16]   R←(R×Y)+L×~Y
       ∇
```

So it turned out that the three best entries all failed exactly one test each – an unexpectedly tough competition indeed.

Morten also supplied the following appropriately named function which ran well over fifty per cent faster (under APL*PLUS PC) than any of the other entries:

```
       ∇ R←FILL QUICKSTEP CTL;⎕IO;STEP;START;END;DIFF;N;MAX;MASK;T;INDEX
[1]      ⍝(0=⎕NC 'FILL')/'FILL←0'
[2]      ⎕IO←0 ◇ START←CTL[;0] ◇ END←CTL[;1] ◇ STEP←CTL[;2]
[3]    ⍝
[4]      MAX←0⌈⌈/N←1+⌊(|DIFF←END-START)÷STEP
[5]      T←N/STEP←STEP××DIFF
[6]      T[¯1+0,+\N]←START-¯1+0,START+(N-1)×STEP
[7]      MASK←N∘.>⍳MAX
[8]      ⎕IO←1 ◇ INDEX←(ρMASK)ρ(,MASK)\⍳ρT
[9]      ⎕IO←0 ◇ R←(FILL,+\T)[INDEX]
       ∇
```

QUICKSTEP was by far the fastest function, but would require modification to change its current bahaviour of ignoring extra elements in the left argument. It also fails the one row matrix of zeroes test.

Competition entries must be written in standard-conforming APL, but we are always interested to see solutions in other APL dialects. Morten included this Sharp APL alternative with his entry:

```
       ∇ R←FILL SAPLSTEP CTL;⎕IO;STEP;START;END;DIFF;N;MAX
[1]      ⍝(0=⎕NC 'FILL')/'FILL←0'
[2]      ⎕IO←0 ◇ START←CTL[;0] ◇ END←CTL[;1] ◇ STEP←CTL[;2]
[3]    ⍝
[4]      MAX←0⌈⌈/N←1+⌊(|DIFF←END-START)÷STEP
[5]      R←(START+¯1↓(STEP××DIFF)×¯1↓⍳N),⍤1⊢(MAX-N)ρ¯1↓FILL
       ∇
```

Congratulations to Neil Mitchison, Morten Kromberg and R H Currie who share the £50 prize money equally. Commendations are also due to Anthony Quas and Heinz Reutersberg.

Surely there must be a better way

Ambi-valent Functions

by David Ziemann

If you have an APL solution that you feel could be improved upon, but you just can't quite see how, then send it in to us. In the other hand, you may have found a new solution to an old problem – why not let other people know about it?

Many APL programmers use interpreters that support ambi-valent functions. An ambi-valent function is one whose valence is not fixed. This usually means that it can be called monadically as well as dyadically. By the way, it's probably better to pronounce 'ambi-valent' with the hyphen in mind; the functions don't feel opposite emotions simultaneously, but rather they demonstrate one of two different combining powers, or valences.

My feelings toward ambi-valent functions are however, definitely ambivalent. They are often used in a way which is likely to lead to code that is difficult to modify, or even worse, that leads to program bugs. This, however is another story, albeit one which I hope to follow up sometime. For the moment, let us say that they should NOT be used for passing 'control information' into a function but rather to allow the function to assume a default left argument. Even this practice is dubious, but. . . .

Anyway, the standard way of testing for the presence or absence of the left argument is by using the 'name-class' system function, ⎕NC. This is demonstrated by the function DIV1, a modified version of DIV, which acts as a cover function for the divide primitive, but which gives a zero whenever division by zero occurs, rather than a DOMAIN ERROR. If the left argument is missing then the value 1 is substituted and the result is a reciprocal.

```
    ∇ Z←A DIV W
[1]   ⍝ Divide <A> by <W> without DOMAIN ERROR
[2]   ⍝ Zeros in <W> give zeros in <Z>
[3]
[4]     Z←Z×A≠W←~Z←W≠0
    ∇

    ∇ Z←A DIV1 W
[1]   ⍝ Divide <A> by <W> without DOMAIN ERROR
[2]   ⍝ Zeros in <W> give zeros in <Z>. Default <A> is 1
[3]
[4]     →(2=⎕NC 'A')/a
[5]     A←1
[6]
[7]   a:
[8]     Z←Z×A≠W←~Z←W≠0
    ∇

      1 2 0 DIV1 2 0 0
0.5 0 0
      DIV1 0 1 2
0 1 0.5
```

If the name-class of the left argument name is 2 then a left argument value was supplied, otherwise it is 0. If you are using a system that supports an APL statement separator then it is possible to tidy this up slightly by coding:

```
    ∇ Z←A DIV2 W
[1]   ⍝ Divide <A> by <W> without DOMAIN ERROR
[2]   ⍝ Zeros in <W> give zeros in <Z>. Default <A> is
[3]
[4]    →(2=⎕NC 'A')/a ◇ A←1
[5]
[6]   a:
[7]    Z←Z×A÷W+~Z←W≠0
    ∇
```

In both cases however, the code is a bit messy, involving a branch arrow, a system function, a label and a pair of parentheses. The next step is to bury the mess inside another function, so that we don't have to look at it all the time. The function DEFAULT assigns its right argument value to the name on the left only if it doesn't already have a value.

```
    ∇ Z←A DIV3 W
[1]   ⍝ Divide <A> by <W> without DOMAIN ERROR
[2]   ⍝ Zeros in <W> give zeros in <Z>. Default <A> is 1
[3]
[4]    'A' DEFAULT 1
[5]    Z←Z×A÷W+~Z←W≠0
    ∇

    ∇ Δ1 DEFAULT Δ2
[1]   ⍝ If name in <Δ1> is not a variable assign it the value <Δ2>
[2]
[3]    ⍎(2≠⎕NC Δ1)/Δ1,'←Δ2'
    ∇
```

Notice that DEFAULT has to use unusual local names in order to reduce the probability of any of them clashing with the calling function's left argument name. Admittedly, the function trades a branch and a label for an execute,but it is hidden away in a single function. This is preferable to the practice of littering code with ever more complex expressions of the form

```
⍎(2≠⎕NC'LEFT')/'LEFT←FOO 2'
```

Apart from the low readability and maintainability of this kind of thing, the call to function FOO would probably not be detected by cross-reference or other workspace documenting programs. Debugging is made easier too, because DEFAULT can be temporarily modified to include your choice of trace or stop expressions.

The DEFAULT function is useful, but if you are lucky enough to be using an APL which has a ⎕ SI system function you can do even better. ⎕ SI typically produces a character vector or matrix representation of the SI stack as it would appear if you use the)SI system command. By examining the SI stack it's possible to determine the name of the calling function, and hence if its left argument name (if any) has a value. The function LARG returns a 1 if its calling function was invoked with a left argument, otherwise a 0 is returned. Because it examines the header line of the calling function's definition, it does not need to have the variable name passed in as an argument.

```
    ∇ Z←A DIV4 W
[1]  ⍝ Divide <A> by <W> without DOMAIN ERROR
[2]  ⍝ Zeros in <W> give zeros in <Z>. Default <A> is 1
[3]
[4]   →LARG/a ◇ A←1
[5]
[6]  a:
[7]   Z←Z×A≠W←~Z←W≠0
    ∇

    ∇ Δ←LARG;⎕IO;Δ1
[1]  ⍝ Return 1 if calling function called with left argument, else 0
[2]
[3]   ⎕IO←0
[4]
[5]  ⍝ Get the name of the calling function
[6]   Δ1←(Δ1⍳'[')↑Δ1←,(1,1↓⍴Δ1)↑ 1 0 ↓Δ1←⎕SI
[7]
[8]  ⍝ Quit if there is no calling function or calling function locked
[9]   →(0∈⍴Δ1←⎕CRL Δ1,'[0]')/Δ←0
[10]
[11] ⍝ Return 1 if the nameclass of the function's left argument is 2
[12]  Δ←2=⎕NC(-(⌽Δ1)⍳'←')↑Δ1←(Δ1⍳' ')↑Δ1
    ∇
```

The system function ⎕ CRL is used to return the character representation of the function header line. This is available in APL*PLUS PC, but users of other systems will have to ⎕ CR the whole function and extract the header line by indexing.

The use of LARG is not recommended because it may encourage 'spaghetti logic'. A better approach is to devolve the assignment of the left argument name into a cover function, as in DEFAULT. In this way the module strength of the function (a system design concept) is not compromised to the same extent. The function LARGDEF (Left ARGument DEFault) implements this idea as follows:

```
    ∇ Z←A DIV5 W
[1]  ⍝ Divide <A> by <W> without DOMAIN ERROR
[2]  ⍝ Zeros in <W> give zeros in <Z>. Default <A> is 1
[3]
[4]   LARGDEF 1
[5]   Z←Z×A≠W←~Z←W≠0
    ∇

    ∇ LARGDEF Δ2;⎕IO;Δ;Δ1
[1]  ⍝ Set calling function left argument to <Δ2> iff its undefined
[2]
[3]   ⎕IO←0
[4]
[5]  ⍝ Get the name of the calling function
[6]   Δ1←(Δ1⍳'[')↑Δ1←,(1,1↓⍴Δ1)↑ 1 0 ↓Δ1←⎕SI
[7]
[8]  ⍝ End if no calling function or function locked
[9]   →(0∈⍴Δ1←⎕CRL Δ1,'[0]')/0
[10]
[11] ⍝ Get the name of the calling function's left argument
[12]  Δ←(-(⌽Δ1)⍳'←')↑Δ1←(Δ1⍳' ')↑Δ1
[13]
[14] ⍝ Assign value iff calling function's left arg. is undefined
[15]  ⍎(0=⎕NC Δ)/Δ,'←Δ2'
    ∇
```

Now we have a function which can be safely used to provide a default value for a funtion left argument name, and without the visible use of branching, labels, execute, parenthesese or quote marks. LARGDEF will have no effect if its calling function definition is not dyadic or if the SI stack is clear.

No reference to the name of the left argument is made in the application function, so the approach is less liable to bugs resulting from program modifications. For example, if you later wanted to rename your function left argument, you could do so with less chance of introducing a program bug.

Can you see why the phrase '0-equal' is used rather than '2-not-equal' in the last line of LARGDEF?

Please note that the function DIV has only been used as an example to demonstrate these techniques; utilities like LARGDEF would be more usefully employed in application functions rather than common APL utilities.

APL Trivia

Funny Dates

compiled by Dave Ziemann

First, thanks are due to John Searle from Sydney, Australia, who was originally responsible for the 'Meaning of life, the universe and everything' expression, which evaluates to 42. What will John's next numerological submission be?

But now to the dates. The following expressions were executed under APL2 release 2, running in a TSO environment:

```
      NAMES←¯11 3↑⍎TSO'APL PDSI ''VSS.NAMES.AP2TN011'''
      ⎕←NAMES←⊂[2]NAMES
 ATR CAN CTK CTN DAN FED KTC PFA RTA SAN SVI
      (⊂3 11) ⎕NA¨NAMES
1 1 1 1 1 1 1 1 1 1
      ⊃[2] 2 ⎕AT¨NAMES
1985 4 1 12 0 0 0
1985 4 1 12 0 0 0
1985 4 1 12 0 0 0
1985 4 1 12 0 0 0
1985 4 1 12 0 0 0
1985 4 1 12 0 0 0
1985 4 1 12 0 0 0
1985 4 1 12 0 0 0
1985 4 1 12 0 0 0
1985 4 1 12 0 0 0
```

The ⎕ AT system function can be used with a left argument of 2 to discover the date and time that any defined function was saved. After defining the APL2 intrinsic functions supplied with the APL2 product, the 2-⎕ AT is applied to each name. As you can see above, it turns out that they were all created exactly at the end of April fools day, 1985! Can this be mere coincidence? Thanks to David Piper for pointing out this curious behaviour.

It makes sense to try 2-⎕ AT with primitive APL2 functions as well, but in this case we just get a timestamp of 7 zeroes – not so interesting. At least, in immediate execution mode we do. However, any primitive APL2 function may be an operand to an APL2 user-defined operator, so it's possible to apply ⎕ AT to this operand within the definition of the operator:

```
      ∇QUAD_AT[⎕]∇
    ∇
[0]   RE←(LO QUAD_AT)RA
[1]   RE←RA ⎕AT 'LO'
    ∇

      2 ⎕AT '⍟'
0 0 0 0 0 0 0
      ⍟ QUAD_AT 2
1614 2 7 18 28 18 285
```

The result when the operator is applied to the log primitive appears to be the birth date of Napier, the Scottish mathematician who discovered logarithms! The question is, is the time accurate? Before you rush to try this with other primitive functions, it only seems to produce a non-zero timestamp for the logarithm function. Further developments awaited eagerly.

Introduction to Contributed Articles

See the APL-86 section (page 77) for our main technical article this issue which is a transcript of Alan Graham's APL86 talk entitled 'Idioms and Problem Solving in APL2'. Alan works for IBM at their Santa Teresa laboratory in California. If you are new to APL2 or an old hand, this profound but eminently understandable foray into the uses of the language will be of interest. Many thanks are due to John Sullivan for his accurate transcription of both text and APL from Alan's talk and visual materials.

David Ziemann is Technical Officer on the I-APL committee, and is responsible for the production of the technical specification of the proposed interpreter. The current technical specification is published here for the first time. Please send any technical comments or suggestions directly to the author.

David Piper is fast becoming a regular contributor to VECTOR. This time he shows us how he has used APL2 language features to create a set of efficient QSAM and BDAM file access functions in a TSO environment. The APL2 code from David's offering in the last VECTOR was accidentally omitted, and so we have appended it to the end of his article.

I-APL Technical Specification

Number 1.1 – October 1986

0 Introduction

This document is the technical specification for I-APL, a full function, portable, freely available public domain APL interpreter for small computers.

As the design and development of I-APL progresses, the technical specification will become more detailed. There are a number of issues which have not yet reached conclusion, and as they become concrete, so subsequent specifications will reflect this.

The specification consists of five parts; an overview followed by hardware, language, environment, and exclusion sections. The hardware specification includes details relating to target machines. The language section contains the specification for the I-APL language itself, and in particular how it relates to the text of the Draft International Standard for APL. The last section contains technical information relevant to aspects of the APL environment, as distinct from the language. Finally the exclusion section includes a checklist of features which are not planned for implementation in I-APL.

1 Overview

I-APL is designed to be a standard-conforming portable APL interpreter for small computers. The standard referred to here is the English text for the Draft International Standard for the programming language APL, or DIS 8485, ISO document number ISO/TC 97/SC 22/WG 3/N55.

Furthermore, I-APL is designed with the education market strongly in mind – full standard-conformance and the minimisation of interpreter size are the twin goals which subordinate execution speed and productivity features. Therefore I-APL is not to be considered a commercially viable product. In particular, there will be no file system or full-screen i/o, and the maximum theoretical workspace size will not exceed 64K bytes.

Effort will not be spent on 'smart algorithms'. For example, the naive bubble sort will be used where required, and symbol table searching will be strictly linear. Additionally, the more esoteric algorithms necessary for a conforming implementation (Eg matrix inverse, matrix divide, dyadic transpose and the transcendental, gamma, and binomial functions) will be candidates for 'magic functions', ie they will be implemented using APL itself, in a manner transparent to the user.

The implementation language will be Forth-like, and the generated object code will be a universal intermediate language which is executed by a small machine-code interpreter. Porting will therefore consist of rewriting a small number of routines particular to the processor and environment of each target machine. The development environment will be initially developed on a PC clone. Paul Chapman will develop the implementation language, the APL interpreter and a small compiler, written in C, to tranlsate to the target machine's language.

2 Hardware specification

I-APL will be implemented for use on the following target machines:

Apple II
BBC model B
Commodore 64
CP/M-based micros
IBM PC and clones
Sinclair Spectrum

Every effort will be made to make it as easy as possible for I-APL to be ported to other small machines. We hope to encourage rather than discourage this activity by providing hooks, handles and documentation wherever they are appropriate.

The size of the interpreter will not exceed 30K bytes, and it is hoped that a final size of 25K bytes will be achieved.

3 Language specification

I-APL will include all the defined facilities and implementation-defined facilities required by the ISO standard, and will achieve a very high level of conformance with the standard. This phrase is used not because it is planned to skip over certain features, but because we recognise that in practice complete conformance is unlikely.

The optional facility 'trace and stop control' will be included, but the shared variable protocol and statement separator facilities will not. A number of consistent extensions will also be made.

3.1 Specified facilities

The following is a list of standard-specified facilities currently planned for inclusion in I-APL:

3.1.1 Functions, operators and variables

All primitive functions, operators, system functions and system variables specified as defined facilities in the standard. That is:

Primitive functions

Conjugate	Negative	Signum
Reciprocal	Floor	Ceiling
Exponential	Natural logarithm	Magnitude
Factorial	Pi times	Not
Plus	Minus	Times
Divide	Maximum	Minimum
Power	Logarithm	Residue
Binomial	Circular functions	And
Or	Nand	Nor
Equal	Less than or equals	Less than
Not equal	Greater than or equal	Greater than

Ravel	Index generator	Shape
Reshape	Join (catenate)	
Roll	Grade Up	Grade Down
Reverse (two forms)	Monadic transpose	Execute
Matrix inverse		
Index of	Member of	Deal
Compress (two forms)	Expand (two forms)	Base value
Representation	Rotate (two forms)	Take
Drop	Dyadic transpose	Matrix divide
Indexed reference	Indexed assignment	Monadic format
Dyadic format		

Also, the 'with axis' forms of the following:

Reverse (two forms)	Compress (two forms)	Join
Expand (two forms)	Rotate (two forms)	

Operators

Reduction (two forms)	Scan (two forms)	Outer product
Inner product		

Also, the 'with axis' forms of the following:

Reduction (two forms)	Scan (two forms)

System functions

Time stamp (TS)	Atomic vector (AV)	Delay (DL)
Line counter (LC)	Name class (NC)	Expunge (EX)
Name list (NL) – monadic and dyadic		
Function fix (FX)	Character representation (CR)	

System variables

Comparison tolerance (CT)	Random link (RL)
Print precision (PP)	Index origin (IO)
Latent expression (LX)	

3.1.2 Miscellaneous

Assignment, branch, parentheses, quotes, labels and end of line comments, as specified.

3.1.3 Quad and quote-quad input and output.

3.1.4 Stop and trace

The 'stop and trace control' optional facility for querying and setting function trace and stop control.

3.1.5 System commands

)CLEAR	)COPY	)DROP	)ERASE
)FNS	)LIB	)LOAD	)SAVE
)SI	)SIC	)SINL	)VARS
)WSID			

It will probably not be possible to provide)COPY on machines which are tape cassette rather than disk-based.

3.1.6 Function editor

A del-type line editor for defining and editing functions. Blank lines will be allowed in defined functions.

3.2 Consistent extensions

The following list of consistent extensions is planned for inclusion in I-APL:

3.2.1 Replicate

The left argument of compression will be extended to the domain of positive integers.

3.2.2 QuadCALL

A facility for executing programs written in the machine language of the host computer.

3.2.3 QuadPW

A facility for querying and specifying a printing width parameter.

3.2.4 Scalar extension

Dyadic scalar extension will be less strict than described in the standard, and will be behave as implemented on most APL systems. That is, a single element array of any rank will conform with any other array.

3.2.5 Base value shape requirements

The shape requirements for the arguments of the base value function will be less strict than described in the standard. That is, a unit dimension in the last axis of the left argument or in the first axis of the right argument will be replicated to match the length of the corresponding axis in the other argument

3.2.6)PCOPY

The)PCOPY system command will be provided in disk-based versions of I-APL.

3.2.7 Join on empties

Joining (catenating) two empty arrays of different type will produce an empty result, rather than the error signalled in the standard.

4 Environmental features

The following list of features related to the APL environment is planned for inclusion in I-APL:

4.1 Atomic vector

A 256-element single character atomic vector will be implemented. All the ASCII characters will be included and will appear in their usual positions. A full range of APL special symbols will be included.

4.2 Character representation

There will be no difference between the internal and external representation of characters in the system.

4.3 Provision of APL characters

APL special characters will be provided on at least one target machine. For other target machines which permit a programmable character set, the APL characters will be provided in a form that will easily allow a porter to install the set. The target machines known to include a programmable character set are the BBC model B, the Sinclair Spectrum and the PC (with colour card). The target machines that are not known to include a soft character set are the Apple II, the CP/M-based machines and the PC with monochrome graphics adapter.

4.4 ASCII representation of APL

All the target machines will allow APL to be entered and displayed using an ASCII representation. The details of this are currently being worked out, and a proposal will appear soon. A keyword or mnemonic approach is considered less desirable than a direct transliteration scheme, although this may not be possible.

4.5 Alphabet

The upper-case alphabet will be available for constructing identifiers, comments and character constants. The lowercase alphabet will not, if as expected it is reserved for use in the ASCII transliteration of APL symbols.

4.6 Internationalisation

All I-APL error messages and system command names will be stored in special tables at identified locations so that alternative language versions of I-APL may easily be created.

4.7 Porting and upgrading

Porting hooks will be provided whenver possible to encourage the transfer of I-APL to other machines, and to facilitate the provision of extra features. For example, hooks will be provided to permit the development of full-screen I/O and arbitrary output translation.

4.8 Numeric representation

The internal representation for the numbers has not been finally decided. There are two possibilities; either a single representation will be used for all numbers, or a multiple representation will be used. If a single representation is chosen it will be a single precision floating point representation of either 4 or 5 bytes. If a multiple representation is chosen, a three-fold split is likely; the floating point representation, 2 byte integers and 1 bit booleans. Note that the advantage of the bit booleans would be purely one of space – no boolean processing optimisations are planned.

4.9 Printing

A facility to echo the current session to a printer will be provided, as will the capability to print the contents of a character array.

5 Exclusion

The following list is a checklist of features that are not currently planned for inclusion in the I-APL base product. It is here to indicate that the features were considered for inclusion and rejected rather than not considered at all.

5.1 A filing system

Other than the transparent one required to save workspaces in a library.

5.2 Shared variables
5.3 Graphics support
5.4 QuadPEEK and QuadPOKE
5.5 A statement separator
5.6 Complex numbers

Complex numbers and complex arithmetic will not be implemented. Also, raising a negative number to a fractional power will produce a DOMAIN ERROR, even when a real result is possible.

5.7 Generalised arrays
5.8 Extensions to upgrade and downgrade
5.9 A defined function locking mechanism
5.10)RESET

)SIC will be implemented.

5.11 S-delta and T-delta

The S-delta and T-delta mechanism for setting handling stop and trace vectors will not be implemented. The corresponding system functions will be present.

5.12 Change name class of functions on the stack

It will not be possible to expunge or fix a function that is pendent, waiting or suspended at a lower level than the top of the SI stack.

5.13 Invalid assignments to system variables

An attempt to assign a system variable with a value outside its valid value set will signal a LIMIT ERROR.

5.14)COPY for tape machines

The system commands)COPY and)PCOPY will not be provided for tape-based machines. They will be included in the disk-based versions of I-APL.

5.15 Screen print

The ability to print the current screen contents to a printer will not be provided.

5.16 Program execution of system commands

System commands will not be in the domain of the primitive execute function.

5.17 Workspaces larger than 64K bytes

A theoretical limit of 64K bytes for workspaces will exist, even for machines that could support larger ones.

A Command driven interface for BDAM and QSAM

Auxiliary Processors using APL2 under TSO

by David Piper

1. Introduction

The requirements for the successful use of files from within APL can be summarised as being:

Efficient input/output processing
A robust, consistent and easy to use interface.

The auxiliary processors associated with the QSAM and BDAM access methods (AP111 and AP210 respectively) are more difficult to use, and generally considered to be less robust than other auxiliary processors associated with file handling in APL (e.g. the VSAM AP, AP123).

Increased difficulty of use arises from the protocols associated with the APs. These are not command driven, but rather depend on the order of assignment/use of the shared variables. The initial values of the shared variables are also crucial, since the file to be accessed is opened at the point of sharing the variables, and closed at retraction, rather than by explicit command after sharing. Also note the data shared variable (prefix REC not DAT) must be shared first.

The APs are considered less robust, since the order of assignment/use of the shared variables is crucial, forming the command interface. Misuse of the variables, in the sense of an incorrect order of assignment/use, can cause the auxiliary processors to abend. If this occurs, the APs may be unavailable for the rest of the TSO session. The QSAM AP is especially prone to this type of failure if only a single (data variable) is shared. In this case, even simple errors, such as attempting to continue processing after end of file, may give rise to an abend in the AP.

2. Designing a new interface

The criteria behind the design of an interface to cover the use of the APs are threefold:

Efficiency of file I/O.
Robustness of the interface.
Ease of use for the application developer.

Since the cover functions are intended for use within an application, ease of use in terms of flexibility of command specification etc. is given the lowest priority. Efficiency is given the highest priority so the command interface has as little impact as possible on application performance.

In order to minimise the efficiency impact, each command is made as simple as possible to interpret. Once interpreted, the minimum possible code is used to execute each command.

The range of commands is extended to include 'block' operations to further minimise overheads. During the execution of blocked commands, the command is interpreted once, then executed in a loop, with no more code than would have to be used if shared variables were used directly by the application code.

Further efficiency is gained by avoiding the use of execute for all operations (except the initial opening of a file). The technique used is to fix an access function, with a given name, in which the references to the shared variables are explicitly coded. For a fuller discussion of this technique, see my article in VECTOR 3.2. Robustness is achieved by containing all file operations within the cover functions. This ensures that all uses of the shared variables are executed in the correct order. It also allows a certain amount of error checking to be performed, such as preventing attempts to write data to a file open in read only mode. Error checking at this point also prevents attempts to process files after an end of file condition is received. Thus the major sources of errors within the auxiliary processors are avoided.

Ease of use is improved by removing the need for initialisation of variables before being shared. The open command performs the correct initialisation and performs all checking necessary to ensure the share was successful. From the applications point of view, the complexity of opening the file is reduced to the simple use of a single command. The same can be said of the close command. The cover function takes care of shared variable retraction and checking of return codes.

Ease of use is further enhanced by the use of the same command structure (as far as possible) across both APs. The command structure has been implemented to resemble as closely as possible that implemented for the VSAM auxiliary processor (AP 123). The similarity of the command structures across all three access methods allows file processing to be used far more easily than when making use of a variety of function driven interfaces.

3. Creating/Deleting Access Paths

Before any files can be accessed, the path function has to be created (or LINKED). For QSAM files:

```
RC←ΔQSAM_LINK 'name'    ⍝ <name> is used to customise the
                        ⍝ names of the access path function
                        ⍝ and the shared variables)
```

Commands can then be issued using the path function:

```
RC←QSAM_name 'command'          ⍝ Opening, Closing or reading

RC←data QSAM_name 'command'     ⍝ Writing data
```

When file processing is complete, the path function can be erased. The process of UNLINKING issues a close command in case any files have been left open. Paths are deleted using the UNLINK function:

```
RC←ΔQSAM_unlink 'name'
```

The path function is expunged along with the shared variables.

4. Opening and Closing Files

As already discussed, opening and closing files is one of the most complex operations under APs 111 and 210. First the variables have to be initialised:

```
CTLQSAM←'filename (ctl'
RECQSAM←'filename (mode conv'
```

After this, the variables can be shared – record variable first:

```
SS←111 ⎕SVO 2 7⍴'RECQSAMCTLQSAM'
```

Return codes from the open operation must be checked:

```
RC←CTLQSAM
```

Using the command interface, the above steps are reduced to a single line of code (assuming the path has already been linked):

```
RC←QSAM_path 'Om filename conv'
```

Acceptable values for <Om> – the access mode – are R(ead), W(rite) or U(pdate). BDAM files can additionally be opened for F(ormat) processing. The return code given by the open operation is fully descriptive of any error that may have occurred. The code can be passed to the relevant error message function to obtain a textual description of the error.

When opened for FORMAT processing, the next command given for the file must be the format command:

```
RC←data BDAM_path 'F' nnn          ⍝ <nnn> is the number of records
```

The file is then left open in update mode.

The BDAM file interface ensures that the open/format commands are processed in the correct order without intermediate attempts to read and write to the file.

Closing the file is simply a matter of using the close command. This command retracts the shared variables (thereby closing the file), but leaves the path function intact. This allows further file processing to take place, either in a different access mode, or to another file.

5. Reading/Writing Data

The only significant differences in syntax between the two file access methods are in the commands associated with reading and writing data. The syntax is bound to be different since BDAM offers direct access to records while QSAM offers only sequential access. BDAM also offers sequential access, the syntax for sequentially processing single records is identical for both access methods.

For efficiency of file access, 'pseudo-blocked' access commands are also provided. These read or write a series of records using only one call to the path function. A loop of code within the access function performs the multiple calls to the auxiliary processor. The command is parsed only once, the code loop simply assigning/using the shared variables as required. As soon as a non-zero return code is encountered processing ceases. If a write is being performed, any unwritten records are returned in the data item of the return vector.

To read a single record, the following syntax is used:

```
(RC DA)←QSAM_path 'R'                          ⍝ QSAM

(RC DA)←BDAM_path 'R'                          ⍝ BDAM sequential
(RC DA)←BDAM_path 'R' record_number            ⍝ BDAM direct
```

The data is returned as a nested vector, each item of the vector is a record from the file. Since only a single record is read, the vector has only one item.

To perform a 'blocked' read:

```
(RC DA)←QSAM_path 'R' number_of_records        ⍝ QSAM

(RC DA)←BDAM_path 'R' n1 n2 n3 ....            ⍝ BDAM direct
```

The blocked access terminates as soon as a non-zero return code is encountered. The length of the vector of records (DA) is the same as the number of records read. The return code indicates why the read was abandoned.

To write a single record, the following syntax is used:

```
(RC DA)←data QSAM_path 'W'                     ⍝ QSAM

(RC DA)←data BDAM_path 'W'                     ⍝ BDAM sequential
(RC DA)←data BDAM_path 'W' n1                  ⍝ BDAM direct
```

A data vector is always returned, normally this will be an empty nested vector. If a non-zero return code is generated by the write command, the unwritten record is returned.

To perform a 'blocked' write:

```
(RC DA)←records QSAM_path 'W'                  ⍝ QSAM

(RC DA)←records BDAM_path 'W' n1 n2 n3...      ⍝ BDAM direct
```

When performing a 'blocked' write using BDAM, records are written until:

Records are exhausted in the data vector.
Record numbers are exhausted in the command vector.
A non-zero return code is received.

Any over-written records are returned in the data component of the explicit result. The left argument is a nested vector, each item representing a record to be written. If only a single record is to be written, consisting of a simple vector, this need not be enclosed.

The convention of using a nested vector to contain data records is adopted to enable the use of the VAR conversion option. This option allows APL2 variables of any type, rank or level of nesting to be written to file without conversion. To write a series of such arrays, each is enclosed to form an item of a nested vector which is presented as the left argument.

6. Conclusions

The primary aim of any system of functions covering the use of file access auxiliary processors should be to maintain the highest level of efficiency possible. This is true simply because of the number of times the auxiliary processor is likely to be used, especially if file processing is involved.

The concept of a command driven interface also enables the following advantages to be realised:

Reduction in the number of functions in the workspace.
Implementation of a similar syntax across access methods.

Generating a function containing the shared variable names explicitly allows multiple files to be accessed without the need to continually retract/re-offer shared variables, and removes the need to use execute.

(Editor: David supplied more APL2 code than we have room for here. A listing of the functions produced by the BDAM and QSAM 'LINK' functions follows.)

```
ΔBDAM_LINK 'TESTLINK'

⍝ Generate the function below
0
      ∇BDAM_TESTLINK[⎕]∇
    ∇
[0]   RC←DA BDAM_TESTLINK CMD;CO;LIM;⎕IO
[1]   ⎕IO←1
[2]   →('O'=↑CMD←,CMD)/op
[3]   RC←⎕SVO 2 12ρ'RECbTESTLINKCTLbTESTLINK'
[4]   →(0≠RC←1 22×v/2≠RC)/er
[5]   →('ORWCF'∈↑CMD)/op,rd,wr,cl,fo
[6]   RC←(1 12)''
[7]   →0
[8]  op:RC←'TESTLINK' ΔBDAM_OPEN CMD←1↓CMD
[9]   →(∧/1 13=↑RC)/0
[10]  bdam_TESTLINK←(∧/0=↑RC)/↑CMD
[11]  →0
[12] rd:→(0≠RC←1 15×~bdam_TESTLINK∈'RU')/er
[13]  DA←(LIM←1⌈¯1+ρ,CMD)ρ⊂''
[14]  →((CO←1)=ρ,CMD)/er
[15] r1:CTLbTESTLINK←(1+CO)⊃CMD
[16] er:DA[CO]←RECbTESTLINK
[17]  →(0≠RC←CTLbTESTLINK)/rx
[18]  →(LIM≥CO←CO+1)/r1
[19] rx:DA←(LIM⌊¯1+CO)↑DA
[20]  RC←(⊂ΔBDAM_CODE RC),⊂DA
[21]  →((12≠↑↑RC)∧v/8 12≠↑RC)/0
[22]  →0(BDAM_TESTLINK 'C')
[23] wr:→(0≠RC←1 15×~bdam_TESTLINK∈'WU')/er
[24]  →(0≠RC←1 33×2≠⎕NC 'DA')/er
[25]  LIM←(¯1+ρCMD)⌊ρDA←,ΔEIS DA
[26]  CO←1
[27]  →(1=ρ,CMD)/sw
[28] w1:CTLbTESTLINK←(CO←1)⊃CMD
[29] sw:RECbTESTLINK←CO⊃DA
[30]  →(0≠RC←CTLbTESTLINK)/wx
[31]  →(LIM≥CO←CO+1)/w1
[32] wx:DA←(LIM⌊¯1+CO)↑DA
[33]  RC←(⊂ΔBDAM_CODE RC),⊂DA
[34]  →((12≠↑↑RC)∧v/8 12≠↑RC)/0
[35]  →0(BDAM_TESTLINK 'C')
[36] fo:→(0≠RC←1 15×bdam_TESTLINK≠'F')/er
[37]  →(0≠RC←1 12×2≠ρCMD)/er
[38]  →(0≠RC←1 33×2≠⎕NC 'DA')/er
[39]  CTLbTESTLINK←2⊃CMD
[40]  RECbTESTLINK←DA
[41]  →(v/0≠RC←ΔBDAM_CODE CTLbTESTLINK)/er
[42]  bdam_TESTLINK←'U'
[43]  →0
[44] er:RC←(RC)''
[45]  →0
[46] cl:RC←⎕SVR 2 12ρ'RECbTESTLINKCTLbTESTLINK'
[47]  RC←(1 22×v/2≠RC)''
[48]  bdam_TESTLINK←' '
    ∇ 30.10.1986 15.42.39 (GMT)
```

```
ΔQSAM_LINK 'TESTLINK'

⍝ Generate the function below
0
      ∇QSAM_TESTLINK[⎕]∇
    ∇
[0]   RC←DA QSAM_TESTLINK CMD;CO;LIM;⎕IO
[1]   ⎕IO←1
[2]   →('O'=↑CMD)/op
[3]   RC←⎕SVO 2 12ρ'RECqTESTLINKCTLqTESTLINK'
[4]   →(0≠RC←1 22×v/2≠RC)/er
[5]   →('ORWC'∈↑CMD)/op,rd,wr,cl
[6]   RC←(1 12)''
[7]   →0
[8]  op:RC←'TESTLINK' AQSAM_OPEN CMD←1↓CMD
[9]   →(∧/1 13=↑RC)/0
[10]  qsam_TESTLINK←(∧/0=↑RC)/↑CMD
[11]  →0
[12] rd:→(0≠RC←1 15×~qsam_TESTLINK∈'RU')/er
[13]  →((2=ρCMD)∧1≥≡__CMD)/rb
[14]  CMD←CMD,1
[15] rb:→(v/0≠RC←1 12×0≠↑0ρLIM←2⊃CMD)/er
[16]  →(v/0≠RC←1 12×LIM<CO←1)/er
[17]  DA←(LIM←⌊LIM)ρ⊂''
[18] r1:DA[CO]←⊂RECqTESTLINK
[19]  →(0≠RC←CTLqTESTLINK)/rx
[20]  →(LIM≥CO←CO+1)/r1
[21] rx:DA←(LIM⌊¯1+CO)↑DA
[22]  RC←(⊂ΔQSAM_CODE RC),⊂DA
[23]  →((12≠↑↑RC)∧v/8 12≠↑RC)/0
[24]  →0(QSAM_TESTLINK 'C')
[25] wr:→(0≠RC←1 15×~qsam_TESTLINK∈'WU')/er
[26]  →(0≠RC←1 33×2≠⎕NC 'DA')/er
[27]  LIM←ρDA←,ΔEIS DA
[28]  CO←1
[29] w1:RECqTESTLINK←CO⊃DA
[30]  →(0≠RC←CTLqTESTLINK)/wx
[31]  →(LIM≥CO←CO+1)/w1
[32] wx:DA←(LIM⌊¯1+CO)↑DA
[33]  RC←(⊂ΔQSAM_CODE RC),⊂DA
[34]  →((12≠↑↑RC)∧v/8 12≠↑RC)/0
[35]  →0(QSAM_TESTLINK 'C')
[36] er:RC←(RC)''
[37]  →0
[38] cl:RC←⎕SVR 2 12ρ'RECqTESTLINKCTLqTESTLINK'
[39]  RC←(1 22×v/2≠RC)''
[40]  qsam_TESTLINK←' '
    ∇ 30.10.1986 15.53.40 (GMT)
```

Using □FX to facilitate the use of Auxiliary Processors

An article with the above title, written by David Piper, appeared in the last VECTOR (volume 3, number 2). Unfortunately the accompanying APL2 programs which David supplied were accidentally omitted from the article. We apologise to David and any readers who have been inconvenienced. The code that should have appeared follows:

```
[⍙]∇ ΔVSAM_LINK.3  p: 14  1986-03-20 14.45.43
[ 0]    RC←ΔVSAM_LINK PN;DA;SV;TSO
[ 1]    ⍝ PN: Generate access function for VSAM path <PN> (AP123)
[ 2]    ⍝ PN - Character vector                    path name to be used
[ 3]    ⍝ RC - Numeric scalar                      RC - 0=ok,1=failed
[ 4]     SV←(2 4⍴'CTLvDATv'),(2,⍴PN)⍴PN          ⍝ Create SV names
[ 5]     →(∨/~(□NC SV)∈0 2)/ev                   ⍝ Report name class problems
[ 6]     RC←123 □SVO SV                          ⍝ Share with AP123
[ 7]     →(∨/0=RC)/er                            ⍝ Error: if no offer failed
[ 8]     RC←□SVO SV                              ⍝ Check degree of coupling
[ 9]     →(∨/2≠RC)/er                            ⍝ Error: if not fully coupled
[10]     RC←SV ΔVSAM_GEN PN                      ⍝ Generate access function
[11]     →(0=RC)/0                               ⍝ Exit if fixed
[12]    ev:'INVALID PATH NAME' □ES 2 3           ⍝ Report name class problems
[13]    er:'VSAM NOT AVAILABLE' □ES 1 5          ⍝ Report unable to share


[⍙]∇ ΔVSAM_GEN.3  p: 12  1986-03-07 12.47.18
[ 0]    RC←SV ΔVSAM_GEN FN;CD
[ 1]    ⍝ FN: Generate access function for VSAM file (AP123)
[ 2]    ⍝ SV - Character array          Shared variable names to be used
[ 3]    ⍝ FN - Character vector         File name
[ 4]    ⍝ RC - Numeric scalar           Return code - 0=ok, 1=failed
[ 5]     CD←⊂'RC←DA VSAM_',FN,' CMD'    ⍝ Function called VSAM_<FN>
[ 6]     CD←CD,⊂'→(2=□NC ''DA'')/dy'    ⍝ Check for dyadic use
[ 7]     CD←CD,⊂'DA←''''''              ⍝ Default null data SV
[ 8]     CD←CD,⊂'dy:',SV[2;],'←DA'      ⍝ Assign data SV
[ 9]     CD←CD,⊂SV[1;],'←CMD'           ⍝ Assign control SV
[10]     CD←CD,⊂'RC←',⍕14,SV,' '        ⍝ Return code is CTL DAT
[11]     RC←↑0⍴RC←1 1 0 0 □FX CD        ⍝ Fix the function


[⍙]∇ ΔVSAM_UNLINK.3  p: 9  1986-03-20 14.48.09
[0]     RC←ΔVSAM_UNLINK FN;NA
[1]    ⍝ FN: Destroy access function for VSAM file <FN> (AP123)
[2]    ⍝ FN - Character vector            file name to be closed
[3]    ⍝ RC - Numeric scalar              RC - 0=ok,1=failed
[4]     NA←(2 4⍴'CTLvDATv'),(2,⍴FN)⍴FN    ⍝ SV names
[5]     NA←(NA,' '),[1]'VSAM_',FN         ⍝ Access function name
[6]     →(RC←3≠□NC NA[3;])/0
[7]     RC←⍎NA[3;],' ''C'''               ⍝ VSAM close file
[8]     RC←∨/1≠□EX NA                     ⍝ Expunge the objects



      ΔVSAM_LINK 'TEST'
0



      ∇VSAM_TEST[□0-]∇
[0]     RC←DA VSAM_TEST CMD
[1]     →(2=□NC 'DA')/dy
[2]     DA←''
[3]    dy:DAT∨TEST←DA
[4]     CTL∨TEST←CMD
[5]     RC←CTL∨TEST DAT∨TEST
```

```
      VREPORT1[□0-]
[0]    RC←REPORT1;VSAM_INPUT;CTL;INPUT;DAT;INPUT;DATA
[1]    RC←&VSAM_LINK 'INPUT'                  ⍝ Link to path INPUT
[2]    →(0≠RC)/er                             ⍝ Report error if failed
[3]    RC←⍎VSAM_INPUT 'OR R1INPUT'            ⍝ Open file R1INPUT for Read
[4]    →(∨/0≠RC)/er                           ⍝ Report error on open
[5]    DATA←0 80⍴' '                          ⍝ Fixed length 80 byte data
[6]   st:RC←VSAM_INPUT 'R'                    ⍝ Read a record
[7]    →(∧/8 4=⍎RC)/ef                        ⍝ End of file, no record read
[8]    →(∨/0≠⍎RC)/er                          ⍝ Report error reading file.
[9]    DATA←DATA,[1] 2⊃RC                     ⍝ Join record to data array
[10]   →st                                    ⍝ Get another record
[11]  ef:CREATE&REPORT1 DATA                  ⍝ Do something with data
[12]   RC←0                                   ⍝ All ok, so return 0
[13]   →qt                                    ⍝ Now tidy up files etc.
[14]  er:&VSAM_ERROR RC                       ⍝ Report error
[15]   RC←1                                   ⍝ Exit with bad code
[16]  qt:VSAM_INPUT 'C'                       ⍝ Always try to close file.
[17]   &VSAM_UNLINK 'INPUT'                   ⍝ Destroy link to path INPUT
```

```
[⍵]∇ &VSAM_ERROR.3  p: 49  1986-03-13 10.49.06
[ 0]    MS←&VSAM_ERROR RC;CD;EN
[ 1]    ⍝ FN: Return error message from VSAM processing via AP123
[ 2]    ⍝ RC - Numeric vector              Return code to be analysed
[ 3]    ⍝ MS - Character vector            Message associated with code
[ 4]     →(1=≡RC)/si                  ⍝ Return code is simple vector
[ 5]     RC←⍎RC                       ⍝ First bit only
[ 6]    si:CD←I 12 13 15 16 17 18 19 20 21 22 27 32 33 42 45 48 0 8 116
[ 7]     CD←CD,4 8 16 20 28 32 40 68 96 100 110 116 128 136 152 168 192
[ 8]     CD←((19↑17⍴1),4,17⍴8),[1.5]CD
[ 9]     EN←(CD⍳⌊.=RC)↑1
[10]    st:→st↑EN
[11]     →0 MS←'VSU1001 E Error creating/ending link with VSAM file.'
[12]     →0 MS←'VSU1012 E Invalid command syntax in the CTL.'
[13]     →0 MS←'VSU1013 I Open request against file already open.'
[14]     →0 MS←'VSU1015 E Command not allowed in current open mode.'
[15]     →0 MS←'VSU1016 E Erase on entry sequenced VSAM dataset.'
[16]     →0 MS←'VSU1017 E Key data too long (or too short in RU command).'
[17]     →0 MS←'VSU1018 S VSAM internal error detected (MODCB error).'
[18]     →0 MS←'VSU1019 S VSAM internal error detected (SHOWCB error).'
[19]     →0 MS←'VSU1020 E Data variable does not contain character data.'
[20]     →0 MS←'VSU1021 E Data variable is not of correct length.'
[21]     →0 MS←'VSU1022 E File is not currently open.'
[22]     →0 MS←'VSU1027 S VSAM internal error detected (TESTCB error).'
[23]     →0 MS←'VSU1032 S Insufficient FREE space for I/O areas.'
[24]     →0 MS←'VSU1033 E No data in the DAT SV for a write request.'
[25]     →0 MS←'VSU1042 E A.P. sequencing error (see □SVC).'
[26]     →0 MS←'VSU1045 S VSAM internal error detected (GENCB error).'
[27]     →0 MS←'VSU1048 E Invalid command sequence (eg W before RU).'
[28]     →0 MS←'VSU0000 I Command executed successfully.'
[29]     →0 MS←'VSU0008 W Duplicate keys on file, only the first read.'
[30]     →0 MS←'VSU4116 S File not closed correctly, use AMS VERIFY.'
[31]     →0 MS←'VSU8004 E End of file, or key greater than any on file.'
[32]     →0 MS←'VSU8008 E Duplicate key, record not written.'
[33]     →0 MS←'VSU8016 E Record not found.'
[34]     →0 MS←'VSU8020 E Record in use by another user.'
[35]     →0 MS←'VSU8028 E VSAM dataset is full.'
[36]     →0 MS←'VSU8032 E Invalid relative byte address.'
[37]     →0 MS←'VSU8040 S Insufficient virtual storage.'
[38]     →0 MS←'VSU8088 E Seq. read requested without prior positioning.'
[39]     →0 MS←'VSU8096 E Attempt to change key of record.'
[40]     →0 MS←'VSU8100 E Cannot change record length in non-keyed DS.'
[41]     →0 MS←'VSU8110 E Attempt to open empty file for read/update.'
[42]     →0 MS←'VSU8116 S File not closed correctly, use AMS VERIFY.'
[43]     →0 MS←'VSU8128 E Attempt to open file not properly allocated.'
[44]     →0 MS←'VSU8136 E Insufficient virtual storage.'
[45]     →0 MS←'VSU8152 E Password error.'
[46]     →0 MS←'VSU8168 E Dataset in use by another user.'
[47]     →0 MS←'VSU8192 E Invalid record number in relative record file.'
[48]     →0 MS←'VSU1099 E Unknown return code: ',⍕RC
```

APL FOR BOARD LEVEL SYSTEMS

£9,000 - £20,000

Metapraxis is a management consultancy specialising in corporate financial control. We normally work with the Senior Directors of groups with annual turnover of over £250 M. Our approach helps Directors to interpret the mountain of data which they face using conventional methods of presentation. We have developed two techniques to facilitate this process:

★ **RESOLVE** is the first of a new generation of corporate control systems, and is used in econometric, banking, and project control environments, as well as for financial control of large organisations.

★ **VISION** is a software control system to coordinate Boardroom presentation media, including computer outputs, such as **RESOLVE** and Prestel, alongside 35mm, T.V., videotape and video-conferencing.

We now seek exceptional individuals to join our software development team. You will develop innovative new products, some of which border on the expert system field, as well as enhance our existing systems and provide in-depth technical support to our clients. Working with advanced graphics techniques, you will use a mix of skills in areas such as APL, Assembler, graphics hardware control and on-line data communications.

You should have a good honours degree in a numerate science, and a demonstrable track record of using APL in the implementation of complex commercial projects, in mainframe and/or micro environments.

This is a unique opportunity to influence and share in the growth of a new industry. Please send relevant career details to:

David Preedy
Development Director

Metapraxis Limited, Hanover House, Coombe Road, Kingston-upon-Thames, Surrey, KT2 7AH.

The British APL Association

Public Domain Software Library

The BAA Public Domain Software Library is now up and running. The library will be managed by the BAA as a non profit-making service for the APL community worldwide, although a discount is offered to BAA members. At the end of April 1986 the scheme will be re-evaluated and changes may be made to its operation.

The library catalogue will be printed regularly in VECTOR, and we hope also to run reviews of popular and interesting software.

So, where does the software come from? You guessed it – the PDSL can only work if you send us your software. The DOS format disk is used as the exchange medium, although this does not mean that the software has to run on the PC. We encourage mainframe users also to share their VS APL, APL2, Sharp APL, APL*PLUS (and any other) utilities or even complete systems. We make absolutely no restrictions on the target machine – provided you can download the software onto a disk, then the library can accept it.

Please think back over the last year or two. That little APL system you developed, those utility workspaces or even that database. It doesn't even have to be APL software, although we do stipulate that it should be 'of interest' to the APL user, programmer, educator or student. When you have something to send us, fill in the PDSL Submission Form (overleaf) and send it off with the software. Before making a submission, remember that if you are not the software owner you should first get permission from that person. You must sign the submission form in order to allow us to distribute the software on your behalf.

Although donors may not sell submitted software to the library, we have taken the decision to permit donation of free demonstration disks, that is software that provides a taste of a commercial product. Software operating under the 'shareware' concept will likewise be allowed, provided that this is explicitly stated on the submission form. At this early stage of the service we do not undertake to distribute paper documentation or any other non-disk materials.

If you are ordering software (use the PDSL Order Form) please understand that we can make no claims or promises whatsoever regarding the software, but we will endeavour to ensure that contributions behave as described by their donors. Furthermore, we cannot accept responsibility for any damage or legal liability caused by using library software. If you do have any positive or negative comments though, let us know and we will take appropriate action.

Finally, please help us get this worthwhile project off the ground by sending us some software, earlier rather than later. You never know, you might even win the prize for donating the most popular disk.

British APL Association Public Domain Software Library
SOFTWARE SUBMISSION FORM

Please copy and fill in this form for EACH disk you submit. If you have any difficulties, see the 'SOFTWARE SUBMISSION INSTRUCTIONS' form. Details correspondong to items flagged (*) will not be made publically available, but are for our records only.

Use BLOCK CAPITALS for all items except numbers 6 and 16.

0. Submission date:_____

1. Name of donor( ):_____ 2. Daytime phone(*):_____

3. FULL address(*):_____

_____

4. Mailbox codes(*): IPSA:_____ STSC:_____ IBM:_____

5. Disk title:_____

6. Brief description of disk contents:_____

_____

_____

_____

7. List target machines:_____

8. List additional software required: _____

9. Indicate special hardware requirements:_____

10. Indicate special software requirements:_____

11. Is sufficient documentation provided on the disk?(Y/N):_____

12. List titles of any paper documentation included with you submission:_

_____

13. Is this documentation, or any other, available to users upon application to you?(Y/N):_____ Please give details:_____

_____

14. Does the disk include any form of payment request from users of the software?(Y/N):_____

15. If the target machine is not a DOS-based PC, does the disk include instructions for transfer to the target machine?(Y/N):_____

British APL Association Public Domain Software Library
SOFTWARE SUBMISSION FORM (continued)

16. File names and descriptions. This information will be made publically available in the software library catalogue. Please save us some work by including these details on a file named <CAT> (no file extension) on your submitted disk. Please enter these details for all files on the disk, except <CAT> itself. You can affix a print of <CAT> below.

FILE NAME	EXT	SHORT DESCRIPTION

17. Total space occupied by all files on the disk, in Kbytes:_____

18. Please use any extra space above for any comments you wish to make.

19. Your signature is necessary. It declares your legal right to make the disk freely available for copying and use, and grants the British APL Association a similar right.

Signature:_____

British APL Association Public Domain Software Library
SOFTWARE SUBMISSION INSTRUCTIONS

The numbered points on this sheet of instructions correspond to items on the SOFTWARE SUBMISSION FORM, which must be filled in and included with with each disk you submit.

1. The donor's name will be made publically available as a part of the software library catalogue. If you wish to make an anonymous donation, please supply your name anyway, but place an asterisk within the parentheses.

2. Please supply a daytime telephone number at which you can usually be contacted. This will not normally be used, and will not be released.

3. The address will not be released. If you wish to make your address available it must be included in an appropriate file on the disk.

4. Please supply any electronic mailbox codes by which you may be contacted. They will not be released.

5. The disk title should be a single short sentence or list by which the disk can be uniquely identified, and which expresses the purpose of the disk.

6. Please supply a short description of the overall purpose of the disk, which may be used as a part of the software library catalogue.

7. List target machines. PC DOS format diskettes are the exchange medium for library software. NO restrictions are made on the final destination of software. Therefore appropriate responses might be: 'IBM PC', 'PC and clones', 'Mainframe with APL2' or 'Any machine running Sharp APL'.

8. List additional software required to use the disk. For example, specifiy the APL interpreter needed if the disk contains APL workspaces.

9. Indicate special hardware requirements. For example: '640K memory needed', 'Hercules graphics board needed' or 'Math co-processor recommended' etc.

10. Indicate special software requirements. For example: 'APL*PLUS PC release 6 needed', 'APL2 release 2 needed' or 'Best results with DOS 3' etc.

11. Indicate if suitable documentation is provided on the disk. A positive response indicates that documentation exists on the disk or that none is necessary. If you are supplying any APL workspaces, you are strongly encouraged to provide variables containing summary descriptions of the primary functions.

12. The British APL Association does NOT at this time undertake to distribute or otherwise make available any paper documentation, or other non-disk materials. You may however choose to submit such material for review purposes - we plan to review library software periodically in VECTOR, the journal of the BAA.

13. Indicate if documentation or other non-disk materials is available to the software user directly from yourself. If so, you must remember to include the details and a contact address ON THE DISK itself.

British APL Association Public Domain Software Library
SOFTWARE SUBMISSION INSTRUCTIONS (continued)

14. Does the disk include a request for payment? If your response is YES, the software library catalogue will indicate that the software is 'USER SUPPORTED'. This warns potential users that the author requires payment under certain specified conditions, eg for privileges available to 'registered' users of the software, such as documentation or future releases.

15. If the target environment is NOT a PC or PC clone running DOS, you are advised and encouraged to provide the necessary instructions to help the user to move the software from such a machine to the target environment.

16. Please specify the file name, extension and short description of EACH file on the disk. To ensure accuracy and to save us extra work, please enter these details into a file named <CAT>, with no file extension, on the submitted disk. You can then print the file <CAT> and affix to the submission form.

    The preferred format for the <CAT> file is 'filename/extn//desc' where the / represents a space, filename as an 8 character filename, extn is a 3 character file extension and desc is a 66 character description. Use a new-line character to delimit these entries.

    If you are submitting APL workspaces, you are strongly encouraged to include variables containing summary descriptions of the primary functions.

17. Specify the approximate amount of space that the disk files occupy.

18. Signature. A submission cannot be accepted without the signature of the software donor.

British APL Association Public Domain Software Library

***** ORDER FORM *****

To: The BAA Public Domain Software Library
c/o David Ziemann
Flat 3, 63 Queens Crescent
London NW5 4ES
ENGLAND

PLEASE SUPPLY THE DISKS CIRCLED BELOW:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53
54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99	100							

TOTAL DISKS ORDERED (please double-check):_____

_____ library disks at 2 Pounds each :	_____	BAA Member's rate
_____ library disks at 3 Pounds each :	_____	Non Member's rate
I want to join the BAA, and enclose the 10 Pounds annual membership fee :	_____	
Postage and handling:	1.00	
Add 2 Pounds for orders outside the UK:	_____	
Total order - remittance included:	======	

Please make cheques payable to the British APL Association.

Payment can be made in US Dollars - pay 1.5 US Dollars for each Pound.

All orders must include payment - allow 30 days for delivery.

BAA membership year runs from 1 May - 30 April. People joining part way through the year will receive appropriate back numbers of Vector.

Software is accepted by the British APL Association on good faith and we do not vouch for or make any claims regarding donated software. The British APL Association cannot be held responsible for any damage, however caused, by the use or misuse of library software.

Index to Advertisers

All queries regarding advertising in VECTOR should be made to the advertising editor, Steve Lyus, at the following address:

Metapraxis Ltd
Hanover House
Coombe Road
Kingston, KT2 7AH

Tel: 01-541 1696

Advertisements should be submitted in typeset, camera-ready A5 portrait format with a 20mm blank border. Illustrations should be black-and-white photographs or line drawings. Rates are £250 per page. A6 and A7 sizes (at £150 and £75 respectively) are available, subject to layout constraints.

BRITISH APL ASSOCIATION

Membership Application Form

Please read the membership information in the inside front cover of VECTOR before completing this form. Use photocopies of this form for multiple applications. The membership year runs from 1st May – 30th April.

Name: _____

Department: _____

Organisation: _____

Address Line 1: _____

Address Line 2: _____

Address Line 3: _____

Address Line 4: _____

Post or zip code: _____

Country: _____

Telephone Number: _____

Membership category applied for (tick one):	86/87	
Non-voting student membership (UK only)	£ 5	
UK private membership	£ 10	
Overseas private membership	£ 18	$ 27
Airmail supplement (not needed for Europe)	£ 8	$ 12
Corporate membership	£ 85	
Corporate membership Overseas	£140	$210
Sustaining membership	£360	

For student applicants:

Name of course: _____

Name and title of supervisor: _____

Signature of supervisor: _____

PAYMENT

Payment should be enclosed with membership applications in the form of a UK sterling cheque or postal order made payable to "The British APL Association". Corporate or sustaining member applicants should contact the Treasurer in advance if an invoice is required. Please enclose a stamped addressed envelope if you require a receipt.

Send the completed form to the Treasurer at this address:

Mel Chapman, 12 Garden Street, Stafford ST17 4BT, UK.

The British APL Association

The British APL Association is a Specialist Group of the British Computer Society and a member of EuroAPL, an organisation supported by the Commission of the European Communities. It is administered by a Committee of eight officers who are elected by the vote of Association members at the Annual General Meeting. Working groups are also established in areas such as activity planning and journal production. Offers of assistance and involvement with any Association matters are welcomed and should be addressed in the first instance to the Secretary.

1985/86 Committee

Chairman:	Dick Bowman 01-634 7639	CEGB, 85 Park Street, London SE1.
Secretary:	Anthony Camacho 0727(56 from London)-60130	2 Blenheim Road, St. Albans, Herts AL1 4NR.
Treasurer:	Mel Chapman 0785-53511	12 Garden Street, Stafford, ST17 4BT
Activities:	Stan Wilkinson 01-286 7068	26 Leith Mansions Grantully Road London W9 1LQ.
Publicity:	Romilly Cocking 01-493 6172	Cocking & Drury Ltd. 16 Berkeley Street, London W1X 5AE.
Journal Editor:	David Preedy 01-541 1696	Metapraxis Ltd. Hanover House, Coombe Road Kingston KT2 7AH.
Education:	Dick Gray 0476-860483	Horseshoe House, Sproxton, Melton Mowbray, Leicestershire LE14 4QB
Technical:	David Ziemann 01-493 6172	Cocking & Drury Ltd., 16 Berkeley Street, London W1X 5AE

Activities Working Group

Peter Donnelly	0420-87024
Steve Margolis	01-670 7959
Tim Perry	04626-77375
Roy Tallis	01-405 7841
Stan Wilkinson	01-286 7068

Journal Working Group

Jonathan Barman	01-493 6172
Anthony Camacho	0727(56 from London)-60130
Steve Lyus	0272-666961
David Preedy	01-541 1696
Adrian Smith	0904-53071
David Ziemann	01-493 6172

WP disks converted to photosetting by Capella House, Stowmarket (0449) 677663.
Printed in England by Short-Run Press Ltd., Exeter.

Abstract

Introduction