

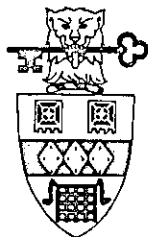
VECTOR

Featuring J

- Hui and Iverson - J questions answered 94
- McIntyre on Hooks & Forks 101
- Chapman on Cross-clocks 124

... plus

- Educational Supplement 17
- Statistics Library News 33
- Crossley on Panel Design 74



*The Journal of the
British APL Association*

A Specialist Group of the British Computer Society

ISSN 0955-1433

Vol.8 No.3 January 1992

Contributions

All contributions to VECTOR may be sent to the Journal Editor at the address on the inside back cover. Letters and articles are welcome 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 accompanied by as much visual material as possible. Unless otherwise specified, each item will be considered for publication as a personal statement by the author. The editor accepts no responsibility for the contents of sustaining members' news, or advertising.

Please supply as much material as possible in machine-readable form, ideally as a simple ASCII text file on an IBM PC compatible diskette (any format). APL code can be accepted as camera-ready copy, or in workspaces from I-APL, APL*PLUS/PC, APL*PLUS II, IBM APL2/PC or Dyalog APL/386.

Except where indicated, items in VECTOR may be freely reprinted with appropriate acknowledgement. Please inform the editor of your intention to re-use material from VECTOR.

Membership Rates 1991-92

Category	Fee	Vectors	Passes
UK Private	£12	1	1
Overseas Private	£20	1	1
(Supplement for Airmail)	£8		
UK Corporate Membership	£100	10	5
Overseas Corporate	£155	10	
Sustaining	£430	50	5
Non-voting Student	£6	1	1

The membership year runs from 1st May to 30th April. Applications for membership should be made to the Administrator using the form on the inside back page of VECTOR. Passes are required for entry to some association events, and for voting at the Annual General Meeting. Applications for student membership will be accepted on a recommendation from the course supervisor. Overseas membership rates cover VECTOR surface mail, and may be paid in sterling, or by Visa or Mastercard at the prevailing exchange rate.

Corporate membership is offered to organisations where APL is in professional use. Corporate members receive 10 copies of VECTOR, and are offered group attendance at association meetings. A contact person must be identified 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 each issue.

Advertising

Advertisements in VECTOR 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 full page, £125 for half-page or less (there is a £75 surcharge per advertisement if spot colour is required).

Deadlines for bookings and copy are given under the Quick-reference Diary. Advertisements should be booked with, and sent to, Gill Smith, whose address is given on the inside back cover.

Corrections to Vector 8.3 January 1992

Quick Reference Diary 1992 [page 5]

There was a small failure of communication and it led to the omission of the Tutorial in *J* meeting from the list.

The 27 March meeting will be *A Tutorial in J* by Donald McIntyre.

Utilities and coding standards for robust systems by Smith and Jordan with an ISO update will be on 1 May.

The 1992 International APL Conference will be in St Petersburg from 6-10 July. To get on the mailing list write to APL 92 Conference Office, PO Box 43, SF-00331 Helsinki, Finland. If you are on email send to the conference office in Russia at APL92@sms.ccas.msk.su or to the conference office in Finland at Juvonen@hekv.m.vnet.ibm.com. We hope Vector 8.4, to be distributed in April, will have abstracts of the papers and details of the workshops and other attractions.

Earlier Dates for Future Issues of VECTOR

We have had to bring forward the dates because we are determined to have Vector Vol 9 No 1 distributed at APL 92 on 6 July 1992 in St Petersburg. All dates for 9.1 should be a fortnight earlier. Sustaining members and advertisers please note that what we don't receive by the end of May (earlier if possible please) will not be printed; there is no slack in this schedule.

We consequently have to ask for 8.4 dates to be brought forward too. Please send all copy by 15 February and advertisements by 28 February (but sooner if possible please).

British APL Association Meetings for 1992 [page 7]

Please note that the meeting listed as 27 March should be listed for 1 May. The meeting on 27 March will be *A Tutorial in J*. Donald McIntyre is coming from Scotland to give the tutorial which will last the whole afternoon. Copies of *J* and the *J* publications will be on sale. We hope to arrange for people to bring and use their portable and notebook PC compatibles to try out for themselves what they learn.

Education Vector is Edited by Alan MAYER

Updating the contents list from 8.2 to 8.3 we omitted to change the Editor of Education Vector; it should, of course already have been changed. Alan Mayer succeeded Alan Sykes as Education Officer at the AGM and has edited Education Vector ever since.

Apologies from the Vector team to all affected by these mistakes.

Anthony Camacho Secretary 20 January 1992



CONTENTS

		Page
Editorial: J is for January	Jonathan Barman	3
APL NEWS		
Quick Reference Diary		5
British APL Association Meetings for 1992		7
APL91 Final Report	Charles Schultz	8
General Correspondence		11
British APL Association News: News from Sustaining Members	Alison Chatterton	12
The Education Vector	Alan Sykes	17
The Random Vector	David Eastwood	33
REVIEWS SECTION		
APL Product Guide	Aiison Chatterton	57
Book Review: Programming in J	Keith Smillie	67
APL and J: Some Benchmarks	Malcolm Rigg	70
RECENT MEETINGS		
Panel Design: An APL Programmer's Toolkit	David Crossley	74
SPECIAL FEATURE: J is for January		
J Questionnaire	Jonathan Barman	90
J Questions and Answers		
Questions	Jonathan Barman	94
Answers	Roger Hui and Ken Iverson	
Hooks and Forks and the Teaching of Elementary Arithmetic	Donald B. McIntyre	101
Cross-clocks in J	Paul Chapman	124
TECHNICAL SECTION		
Hackers Corner: More about VGA Colours	Adrian Smith	132
Technical Correspondence	De Kerf and Spunde	136
Arrays with Style	Adrian Smith	140
Index to Advertisers		143

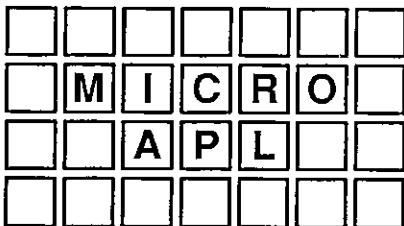
Some things are worth the wait

APL.68000 is not the only APL available for IBM's RISC System/6000. But no other APL running on the RISC System/6000 offers all these features:

- MicroAPL's acclaimed windowing interface
- Seamless integration with X-Windows
- Adherence to OSF/Motif style guidelines
- Very close compatibility with APL2/370
- Binary workspace compatibility across platforms
- High performance through MicroAPL's unique "Portable Assembler"™ technology

APL.68000 for the IBM RISC System/6000

*Fourth to announce.
First to get it right.*



MicroAPL Ltd.
South Bank Technopark
90 London Road
LONDON, SE1 6LN
Telephone: 071 922 8866
Fax: 071 928 1006

Corrections to Vector 8.3 January 1992

Quick Reference Diary 1992 [page 5]

There was a small failure of communication and it led to the omission of the Tutorial in J meeting from the list.

The 27 March meeting will be *A Tutorial in J* by Donald McIntyre.

Utilities and coding standards for robust systems by Smith and Jordan with an ISO update will be on 1 May.

The 1992 International APL Conference will be in St Petersburg from 6-10 July. To get on the mailing list write to APL 92 Conference Office, PO Box 43, SF-00331 Helsinki, Finland. If you are on email send to the conference office in Russia at APL92@sms.ccas.msk.su or to the conference office in Finland at Juvonen@hekv.m.vnet.ibm.com. We hope Vector 8.4, to be distributed in April, will have abstracts of the papers and details of the workshops and other attractions.

Earlier Dates for Future Issues of VECTOR

We have had to bring forward the dates because we are determined to have Vector Vol 9 No 1 distributed at APL 92 on 6 July 1992 in St Petersburg. All dates for 9.1 should be a fortnight earlier. Sustaining members and advertisers please note that what we don't receive by the end of May (earlier if possible please) will not be printed; there is no slack in this schedule.

We consequently have to ask for 8.4 dates to be brought forward too. Please send all copy by 15 February and advertisements by 28 February (but sooner if possible please).

British APL Association Meetings for 1992 [page 7]

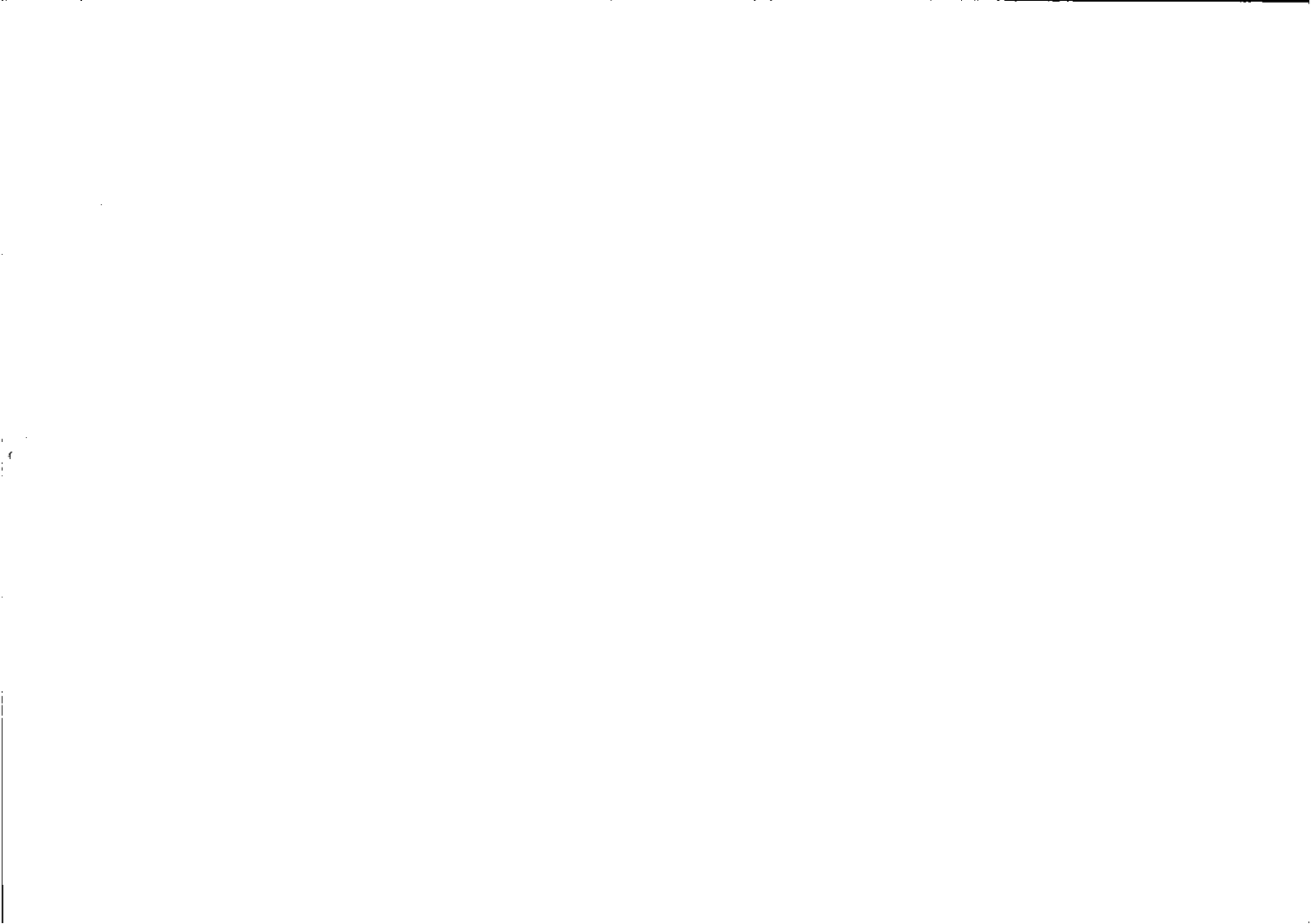
Please note that the meeting listed as 27 March should be listed for 1 May. The meeting on 27 March will be *A Tutorial in J*. Donald McIntyre is coming from Scotland to give the tutorial which will last the whole afternoon. Copies of J and the J publications will be on sale. We hope to arrange for people to bring and use their portable and notebook PC compatibles to try out for themselves what they learn.

Education Vector is Edited by Alan MAYER

Updating the contents list from 8.2 to 8.3 we omitted to change the Editor of Education Vector; it should, of course already have been changed. Alan Mayer succeeded Alan Sykes as Education Officer at the AGM and has edited Education Vector ever since.

Apologies from the Vector team to all affected by these mistakes.

Anthony Camacho Secretary 20 January 1992



Editorial: J is for January

by Jonathan Barman

J is Ken Iverson's new dialect of APL, and is the theme for the January issue of Vector. The issue contains a wide range of material: a tutorial by Donald McIntyre, a real problem solved in J by Paul Chapman, the future of J and hints on programming style by Roger Hui and Ken Iverson, comments from people who have already got J, and comparative timings using the Smith bench marks. All this should help you decide if you want to invest your time in exploring the world of J.

The very first steps learning J are easy, but the going rapidly gets more and more difficult. Really learning the language is extremely hard work, but I think worth the effort. I found it quite nostalgic studying J as I kept being reminded of when I learnt APL from a quick reference card. It was fun discovering the powerful features available and then speculating on how they might be used in some arcane and quite unexpected way.

The definition of the language is the *Dictionary of J*. It is very precise and terse, taking a mere 18 pages to describe every aspect of a large language containing some 180 features. In places it is quite difficult to understand. One feels that each word has to be studied with the greatest care to extract every possible nuance and shade of meaning. However, after a while one begins to appreciate the way in which it is laid out and to enjoy the concise descriptions.

The learning technique suggested in *Programming in J* is to discover for oneself how things work by experimentation at a machine. In general this is good advice, and given plenty of time it would be possible to discover exactly how every aspect of the language works by this means. Unfortunately, time is limited and experimentation can be very time consuming. It would be marvellous to have an extended description, with examples, of each feature of J.

I recognised many features of J from papers which have been published in the APL conference proceedings. In some cases ideas have changed since the original papers, but they were most illuminating as they go into detail on how and why the feature is designed, and give extended examples of their utility. It would seem to be reasonably straightforward for someone to collect all the relevant papers, bring them into line with the current version of J, and to publish them as an amplification of the Dictionary.

The ISI catalogue states that J is particularly intended for use in education and research. It is possible that J could become used general programming when it has settled down and can be linked with other facilities. LinkJ enables C programs to call J and vice versa, so J could be a component of major systems. I could well see myself starting to use J seriously when I can see how to link it to a user interface such as Windows 3 and to a reporting system. Until this has been demonstrated, though, J is for me strictly for play.

J is still in the process of rapid development. New versions seem to come out every few months, though Ken Iverson says that the future pace of development will be slower. The articles in Vector cover versions 2.9 to 3.5x4, and by the time you receive Vector version 4 should be available. It is good to know that users are being listened to, and that when difficulties are experienced the language can be changed quickly, but one wonders why such frequent changes are necessary. Thank goodness that there is no need to worry about upward compatibility. I have seen so many cases in the 4GL world where features have been bolted on, resulting in a mass of strange and inconsistent syntax.

Because J is shareware the cost of a trial copy is easily affordable, and I am sure that many people will want a copy to see what it is like. Vector will continue to cover J and its developments. At some time in the future this could become an important language.

Forthcoming Change of Administrator

Alison has very nearly had enough! Look out for changes in advertisement booking, Product Guide Updates, and the address for membership renewals. Full details will be in Vector 8.4, but current intentions are:

- all membership administration will be transferred to Rowena Small, at 8 Cardigan Road, London E3 5HU (Tel: 081-980 7870)
- all Vector related activity will come to Gill Smith, at APL-385, Brook House, Gilling East, York (Tel: 04393-385)

Watch this space (carefully!)

Quick Reference Diary 1992

Date	Venue	Event
14 Feb 1992	IEE London	BAA Meeting
27 March	IEE London	ISO Progress Report + Utilities and coding standards for robust systems (Smith & Jordan)
1 May	IEE London	BAA Meeting
12 June	IEE London	BAA AGM + 25 years of APL
25 September	IEE London	Vendor Forum
23 October	IEE London	Field trip to Morgan Stanley
27 November	IEE London	BAA Meeting

Starting from February 1991, all British APL Association meetings will be held in the IEE, Savoy Place. Nearest tube outlets: Temple or Embankment.

Meetings are normally held on the 3rd Friday of the month throughout the autumn and spring.

Dates for Future Issues of VECTOR

	Vol.8 No.4	Vol.9 No.1	Vol.9 No.2
Copy date	1st March 92	1st June 92	1st Sept 92
Ad booking	1st March 92	1st June 92	12th Sept 92
Ad Copy	12th March 92	14th June 92	21st Sept 92
Distribution	April 92	July 92	October 92



Cocking & Drury Ltd

**Your sole UK supplier
of the entire APL*PLUS software range**

including:

APL*PLUS PC

APL*PLUS II for the 386

APL*PLUS II for UNIX

APL*PLUS VAX/VMS

APL*PLUS Mainframe

**as well as Run-Time and Run-Time Developer Kits
and all delivered ex-stock!**

**The only dealer authorised to provide maintenance and support for APL*PLUS
All software prices include 12 months free maintenance, with free upgrades (except
APL*PLUS PC).**

**We also offer a wide range of courses in APL and the most experienced APL
consultancy service in the country.**

Whatever your APL needs - call the experts first.

Cocking & Drury Ltd 180 Tottenham Court Road, LONDON W1P 9LE

Phone 071-436 9481 Fax 071-436 0524

British APL Association Meetings for 1992

14 February 1992

1. SEEK, an APL associative knowledge base. Graeme Robertson.
2. Uses and abuses of APL2 for documentation. Ron Wilkes of IBM.
3. Report on the progress developing the APL Statistics Library.
4. Mastering J. A video of Donald McIntyre delivering his paper at APL91.

27 March 1992

1. Report on the progress of the ISO APL language standards, and how this affects the future of APL.
2. Write a system with 10% of the code! Adrian Smith will tell us how a properly constructed utility library can be used to maximum advantage, so that only 10% of the system actually needs to be written from scratch.
3. APL coding standards and methodologies for robust systems: Maurice Jordan.

12 June 1992

1. **Annual General Meeting.**
2. News from SigAPL and their idiom collection, by Dick Bowman. Remember the FinnAPL idiom list? We are expecting great things from the SigAPL idiom list, which should cover a much wider range, and include idioms for many of the dialects of APL.
3. 25 Years of APL. Video of the panel discussion at APL91 where the founders of APL reminisced about events over the last 25 years.

25 September 1992

Vendor Forum. Learn about the latest products and plans from the major vendors of APL - Dyalog APL, IBM, MicroAPL, STSC/Cocking & Drury. Shareware will be represented with IAPL, J and Sharp APL. In previous years the Vendor Forum was held in May. Moving the date to September will enable vendors to report on the announcements that they have made at the previous APL conference.

23 October 1992

Field trip to Morgan Stanley. John Searle has organised a visit to the new Docklands offices of Morgan Stanley, where we can see how three versions of APL are used to create efficient systems. The super fast language 'A' is unique to Morgan Stanley, and they also use Dyalog APL and Sharp APL.

APL91 Final Report

by Charles Schultz

We were glad that so many of you could travel to APL91 in this year of uncertain international events and economic problems. We were gratified to hear many compliments and to see the (mostly) positive reviews in the previous Vector. Of course there were some problems. A couple of years ago, when some of us were first thinking of hosting APL91, I thought "Hmm, I've been to a lot of these conferences, I know what I've liked and not liked - sure, we could do this." I can now report that I, for one, have learned my lesson.

Attendance

The final full registration was about 370, representing 22 countries. We had an additional 75 spouse and child registrations. About 40 attended the CASE Sunday tutorial, and 30 attended the Sunday introduction to APL tutorial.

Outreach Publicity

We undertook several efforts to reach outside the APL community. We sent press releases, calendar notices, and other materials to 700 magazines and newspapers. In California, we sent materials to 200 scientific centers, such as university and corporate research institutes. We sent introductory tutorial fliers to 350 high schools in the Bay Area and 250 college computer science departments.

We solicited vendor interest among producers of word processors to handle APL features and gain access to an international community, of terminal emulators to handle APL communications, and of hardware that runs APL. We even invited a list of Silicon Valley celebrities to the closing picnic. Steve Jobs did give his personal regrets by telephone; John Sculley had his secretary do it for him.

Author Kit and Font Table

We were ambitious about contributing to momentum on author tools, since we view the APL publishing problem as one of the reasons it is so difficult to promote APL. We requested author tool proposals from several individuals. The few tool proposals we received were interesting, but not in an "install and go" form that we could pass on. All had dependencies on computer system, word processing software, and APL system. The only tool we were able to publish was

an IBM BookMaster style file for the paper format (BookMaster contains standard APL features).

We requested information from all authors on the word processing systems they were using (very few provided this). We solicited WordPerfect and Microsoft as the leading word processing vendors to take a vendor booth and to consider providing an APL font to capture an international market of APL programmers. We included references to some PostScript font work that has been done. Jan Engel, the proceedings editor, is now corresponding with a contact at WordPerfect. IBM is now working on a PostScript APL font for OS/2 Communications Manager.

The Font Table at the conference was conceived as a result of all the troubles we had with publishing APL for the proceedings. We hope this effort will continue in and outside future conferences towards some distinct and "pluggable" solutions.

APL91 Museum (*George L. Mendonsa*)

The APL Museum at the APL91 conference was set up ad hoc toward the end of the conference preparation period, as other more pressing items fell into place. George Mendonsa, with the assistance of Jan Engel, put the Museum together. (Because of its informal nature and set-up a more appropriate term might have been "Scrapbook" instead of "Museum".)

The museum was initiated to take advantage of the historic nature of the conference, a desire to begin to identify APL material of potential interest, and to find out the nature of the interest in such an endeavour. Calls for contributions to the museum were made to likely sources in part suggested by the APL91 committee. These people in turn suggested other sources. Contributors were asked to provide materials in one of 2 categories:

1. Items of Historical interest in the evolution of the APL milieu.
2. Related items of APL interest which were not necessarily historical.

We had about 25 to 30 contributors who included APL92, Linda Alvord, Ev Alan, Paul Berry, Larry Breed, Jim Brown, Carl Cheney, Ed Cherlin, Jan Engel, Adin Falkoff, Garth Foster, Vern Griffith, Interprocess, Curtis Jones, Ken Iverson, Gene McDonnell, Don Mattern, George Mendonsa, Don Peter, Marilyn Pritchard, Lynn Shaw, Bob Stephan, STSC, Rex Swain, and Joey Tuttle. The Museum was coordinated by George Mendonsa. Larry Breed had an extensive collection of items and was available at the Museum for much of the time. Lew Robinson, Jan Engel, Bob Stephan, Carl Cheney also acted as Docents at various times.

Items on display included buttons, books, papers, documents, and items from previous APL conferences such as The APL jig-saw puzzle, APL Tool-kit, Apple Blossom Time record, 1130 keyboard, Proceedings, T-shirts, etc. Larry provided the listing of the very first logon to APL. This was duplicated and handed out at the conference. Linda Alvord had sent The APL Genealogy project report (begun at APL88) which was displayed and added to at the conference by over 100 individuals. Don Mattern demonstrated his Ampere WS-1 APL laptop. Don Peter set up an on going demo of the IBM 5110. We also had a book as a memorial to Alan Perlis, Mike Montalbano, and Bill Bergquist which included some of their works. Interprocess had a collection of 40 APL buttons.

The Museum was well attended with over one half of the conference attendees stopping by. As indicated there does seem to be a interest such a facility at conferences.

Let us be very clear: this was a one time experimental event. The museum ceased to exist at the end of the conference. All items with a few exceptions were picked up by the contributors when the Museum closed. There was an attempt at the conference to determine who was interested in creating some such repository. Although a few expressed interest, no formal funded organization is in place. If you are interested in such a function, perhaps you could volunteer to your national APL organization.

It may be difficult to have an actual museum, but an on-going Ad Hoc portable museum (or scrap-book) could be put together for future conferences (as was demonstrated by this one). But even more important would be a repository. This would be a listing of items of interest that people had that: 1. they would like to contribute to a museum or 2. they would retain themselves but would like other people to know about.

Further, a repository could be created for each country and perhaps maintained and exchanged through the national APL organizations. Also, the genealogy project could continue as an ongoing activity in each Museum.

General Correspondence

From: Phil Last

2nd December 91

Everybody is welcome, whether they are a member or not
 LENGTH ERROR

Everybody is welcome, whether they are a member or not
 ^

Scalar extension is not implemented in English. Grammatically correct would be:

Everyone is welcome, whether one is a member or not.

Everyone is welcome, whether he is a member or not.

All are welcome, whether they are members or not.

One could argue that the Regal usurpation of the pronoun 'one', albeit that its proper use (c.f. Fr 'on', Ger 'man') is Not in place of 'I', renders this usage obsolete.

He could argue that the sexist connotations associated with the pronoun 'he' render this usage controversial.

She could elide the pronoun and verb, giving:

Everyone is welcome, whether a member or not.

The qualification is tautological in any case. In fact, it hardly qualifies as a qualification:

$\wedge / \omega \leftrightarrow \sim \vee / \sim \omega$

In other words if we take 'Everyone' as our universe of discourse, then it must subsume:

'whether a murderer of the English Language or not'

'whether the Editor of Vector or not'

... and the union of any other set with its complement.

This leaves us with a fairly concise (as is usual in APL contexts):

Everyone is welcome

... or to revert to the plural:

All are welcome

News from Sustaining Members

Compiled by Alison Chatterton

Cocking & Drury

There has been a small change in ownership at Cocking & Drury. Peter Day has purchased a majority stake in the subsidiary Cocking & Drury (Software) Ltd. He is now responsible for all aspects of the business to do with software sales (for STSC - APL*PLUS and Statgraphics; and Digitaltalk - the Smalltalk/V range).

Romilly remains Managing Director of Cocking & Drury Ltd and responsible for consultancy and training work - again both APL*PLUS and Smalltalk/V.

The two companies continue to operate from the same premises, with the same telephone numbers etc. Romilly will be assisting Peter in his field, and vice-versa. Cocking & Drury Ltd still holds a substantial minority stake in the Software company.

This change has brought about a considerable injection of capital into Cocking & Drury, helping to keep the wolf well away from the door during the current recession. Customers can, therefore, continue to rely on the high quality of service and support they have been used to in the past, and we at Cocking & Drury will be continuing to invest in our future in both APL*PLUS and Smalltalk.

Free copies of the upgrade to APL*PLUS II for the 386 Version 3.5 were sent to all customers on maintenance during November. This upgrade gives the ability to run as a non Windows application under Windows. Version 4 is currently in Beta test and will include an interface to the Paradox engine and other major additions. Release is expected late February.

STSC are concentrating their development efforts on the APL*PLUS II systems for 386/486 PC's and for Unix. We expect to see full Windows support for the PC product in the not too distant future and, with the inclusion of a Dynamic Link Library, the ability to use APL*PLUS as the co-ordinator of a suite of software or as a calculating engine with other products, like Smalltalk/V, as the front-end.

It is rewarding to see more and more customers moving their applications from the APL*PLUS PC environment to the II system and reaping the rewards of the greater speed and the large workspaces without significant recoding.

STSC have also applied the 32-bit technology to their Statistical graphics package Statgraphics. The launch of Statgraphics Plus removes the limits on data-set sizes that existed in the original product and make an already popular package an outright winner for data analysis.

The next issue of our magazine 'Upgrade' will be distributed in January, if you don't think you are on our distribution list - drop us a line. We are also holding a Showcase event in our offices on Thursday February 13th. If you would like to come along to see the latest developments in the APL*PLUS (and Smalltalk/V) worlds - and demonstrations of applications using both technologies; give us a ring.

MicroAPL Ltd

MicroAPL is pleased to announce that we have been appointed by IBM (UK) as the 'Software Sales Representative' for IBM's APL2 mainframe and PC software. MicroAPL are one of the first of a new type of sales representative to be appointed by IBM to help to promote specialist software products such as APL and we will be acting in cooperation with IBM's regular sales force and as part of that sales force.

Starting through eight pilot branches, we shall be visiting major APL accounts and discussing the ways in which APL is used and to promote awareness of IBM's range of APL software. IBM (in its recent APL2 Version 2 announcement) has reaffirmed its support and enthusiasm for APL2 - '... Customers should continue to choose APL2 as a language when it is appropriate, and should not plan to convert existing programs from APL2 to another language solely because APL2 is not a designated SAA language....'. Here at MicroAPL we shall be vigorously supporting this approach and at the same time seeking to find and highlight new areas of suitability for APL in partnership with other software tools. We shall also be making sure that APL2 users are given access to a full range of support, training and consultancy services.

Our appointment coincides with the announcement of Version 2 of APL2 (mainframe) which brings important new features for users of APL2. These include the ability to process files as variables without bringing the file into the workspace (AP12); improved workspace storage management and page release performance; the return of AP124!; major new additions to facilitate cooperative processing including cross-system shared variables, support for TCP/IP (AP119) and a remote session manager; and a range of enhancements and additions to existing facilities. The full details are too numerous to include here, but if readers

have not seen the APL2 Version 2 announcements, please contact us for full details. APL2 Version 2 has a planned availability date of March 1992.

David Eastwood is currently managing MicroAPL's relationship with IBM, but other members of staff will be involved as the project picks up momentum.

Turning now to our own products, APL.68000/X is now nearing the end of its beta test program and represents our most advanced version of APL.68000. We have added a wide range of significant new features for the RISC System/6000, many of which will move onto other versions of APL.68000 (such as the Macintosh version). The bulk of our activities with APL.68000/X have been to produce an application which is genuinely compliant with the user interface standards prevalent on the RISC System/6000. At the APL level, we have included on-line Help for the first time (by default this is the full APL.68000/X manual but users can add their own Help files) and introduced new facilities to allow users to edit, save and load their APL.68000/X session details. We have also added a full set of high level tools to permit APL programmers to produce sophisticated windowing applications using a range of APL facilities rather than low level C routines. We have at last formalised our interface to native files by implementing a series of native file system functions (`□NREAD` etc.) which are compatible with other implementations of these system functions.

Impetus Limited

We are now reaching the end of an extensive development of *Impetus* which is a Corporate Modelling and Reporting Product. This has taken nearly two years and has involved the production of an additional 600,000 bytes of APL (*Impetus* is now about 1.4 megabytes of code) and the writing and production of over 800 pages of documentation. We have also produced nearly half a megabyte of on-line help screens.

The development has been in three main areas. Firstly, we have built extensive business graphics which is tightly integrated into the rest of *Impetus*. Secondly we have added menu facilities to enable the user to browse around the data in the project and to browse around the facilities in *Impetus*. Thirdly, we have removed any need for the user to understand APL.

This last change was especially important as a user sits in APL Command Mode and invokes *Impetus* functions. The user is not encapsulated in code which limits the facilities. In a sense *Impetus* is an open architecture system and the user can add new functions written in APL. In the past this was necessary for successful operation but this is no longer the case. The *Impetus* language is now rich

enough to make the use of APL unnecessary, but it does not preclude the APL speaker from using APL if that is the user's desire.

We have also changed our trading style and have dropped APL from our company name and are just Impetus Limited.

APL People Limited

I'd like to take this opportunity to wish everybody a Happy (and prosperous) New Year! 1991 wasn't an easy year for anybody, companies and individuals alike, and the APL Community has not been immune to the effects of the recession (APLers can be made redundant like anybody else). I was pleased that I was able to help a number of people to find employment in difficult times. Don't give up hope, those of you who are still looking, - things are bound to improve this year. I am convinced that the worst is over and that demand for people with APL skills will increase this year. Anyone wishing to discuss the current state of the employment market and what his or her prospects might be in 1992, is welcome to call me (see our advertisement for the telephone numbers).



APL PEOPLE

Serving the International APL Community

Consultancy advice and assistance with all aspects of APL systems development

Recruitment unique placement service for companies and individuals

Software Competitive prices on all APL software tools and interpreters

Contact: Jill Moss, APL People Ltd.,
The Old Malthouse, Clarence Street,
Bath BA1 5NS

Tel: 0225 462602 during office hours
0225 333618 evenings and weekends

Dyadic Systems Limited

Dyadic is pleased to confirm a hitherto unproven feature of Dyalog APL; namely its ability to access DB2 databases held on an IBM mainframe using the Dyalog APL interface to Oracle. This facility was proved in the course of installing Dyalog APL for a large French bank during December. The configuration for the test was in itself rather unusual. Dyalog APL/X was installed on an IBM RS/6000 in Paris together with a copy of Oracle. The RS/6000 was networked to a DEC VAX located elsewhere in France. The French VAX was connected by satellite to a second VAX in California, which was itself networked to an IBM mainframe running DB2. Finally, just to make things interesting, the Dyalog APL user was sitting, not on the RS/6000, but on a Sun workstation connected to it by ethernet. As Dyalog APL/X can be operated from anything that runs X, this was not in itself a problem. Nevertheless, it was with some trepidation that our user first typed :

```
SQL 'SELECT * FROM emp@ibm1'
```

... but lo and behold the correct result appeared. Becoming bolder, the team tried database "joins" between DB2 tables on their remote mainframe, and local Oracle tables on the RS/6000. Again, everything worked just as the Oracle salesman said it would. So if you want to access DB2 databases from your Unix workstation, all you need is Dyalog APL, a copy of Oracle, and a bit of wire!

Progress with the development of Dyalog APL/W is good, although somewhat slower than planned. Dyadic is one of the first companies anywhere in the world to develop a 32-bit Windows 3 application, and there have been many new technical problems to resolve. The product is nearing completion and the company now expects to ship a test release at the end of January. Dyadic is especially pleased with the interface to the Windows 3.0 GUI which has been designed not just for Windows, but also for OSF/Motif, Open Look, etc. Dyadic's intention is to provide the same APL->GUI programmer interface on all "windows" environments, thus ensuring true portability of Dyalog APL applications.

THE EDUCATION VECTOR

January 1992

Editor Alan Mayer

This Education Vector has been reprinted from VECTOR Vol.8 No.3. VECTOR is the Quarterly Journal of the British APL Association. For more information about the British APL Association, please contact: Anthony Camacho, 2 Blenheim Rd, St Albans, Herts AL1 4NR Tel: 0727 860130.

Contents

Editorial	Alan Mayer	18
Letters		19
An Instruction Sheet for Introducing APL to Lay Audiences (Teachers)	Zdenek V. Jizba	19
Curve-fitting in APL	David Appleton	23
How to produce APL print on some Star printers with I-APL	Camacho & Goodman	29

Dr Alan Mayer
European Business Management School
University College of Swansea
Singleton Park
Swansea SA2 8PP Wales, UK

Ysgol Rheolaeth Busnes Ewropeaidd
Parc Singleton,
Abertawe SA2 8PP

Tel. 0792 205678 Ext.4274 Fax: 0792 295626 JANET: MAYER@UK.AC.SWAN.MS

Editorial

Welcome to Education Vector, and Happy New Year! 1992 has long been heralded as the year when we will discover the advantages of being European. As I write, discussions are going on in Maastricht which may bring great changes to the structure of government in all the countries of Western Europe. Even more profound changes are taking place further afield. Like many APLers I believe that APL has a significant role to play in bringing people of different languages and cultures together. It is a language that crosses all barriers and frontiers. If 1991 is anything to judge by, the pace of change will be bewildering. When we made tentative plans to hold APL92 in Leningrad, USSR, we did not expect to find the venue renamed St Petersburg - and recent reports suggest that the USSR itself will no longer exist by the time the conference takes place. Be that as it may, bringing together the world-wide APL community is an important aim, and our own efforts to "spread the word" in Education Vector should be designed to contribute to that aim.

While many readers do not yet have access to second generation interpreters, I feel that one of our functions should be to explore the possibilities of the very powerful extensions to the language that are now available. I am pleased to include an article by David Appleton in which he makes elegant use of multiple assignments in the field of curve-fitting. I for one would like to see more contributions telling us how best to make use of all these new toys!

Most of us have little functions sitting in odd workspaces, written to meet a particular need and then forgotten. I have several designed to introduce children to computers, and perhaps to help them learn other things as well. If you are nodding your head in agreement now, dig some of yours out and send them to me. My little contribution, to start the ball rolling, is a little program to help with the problem of multiplication (I carefully avoid the word "tables" - are they mandatory or illegal at the moment?)

```
[0]  MULTIPLY;ANS;I;RV;T;TARGET
[1]  RV←? 10 2 ρ12
[2]  ANS←:0
[3]  I←1
[4]  T← 0 60 60 1¯1+¯4+□TS
[5]  TARGET←×/RV
[6]  loop:'What is ',(⊖RV[I;1]),'×',(⊖RV[I;2]),'? '
[7]  ANS←ANS,1+□
[8]  →(10≥I+I+1)/loop
[9]  'You scored ',(⊖+/ANS=TARGET),' out of 10'
[10] 'in ',(⊖(0 60 60 1¯1+¯4+□TS)-T),' seconds'
```

Just type *MULTIPLY* and answer the questions as quickly as you can! I have tried to use "universal" APL - if you have to modify it to make it work, please write to me about it.

Letters

From: Neil Sheldon, Head of Mathematics, Manchester Grammar School.

In the *Education Vector* for October, Professor Tony O'Hagan makes out a heuristic case for a version of the trapezium rule rather than Simpson's rule when integrating over more than three points. His argument is appealing but, unfortunately, wrong.

The rule which Professor O'Hagan advocates is equivalent to a pair of Simpson's rule estimates across the body of the integral together with "a bit of trapezium rule on the first and last intervals". It is these bits of trapezium rule which cause the problems. They have errors proportional to h^3 , (where h is the strip width) while the Simpson's rules have errors proportional to h^4 . It follows that, for small enough h , the errors in the bits of trapezium rule must dominate, and hence that Professor O'Hagan's rule must perform less well than Simpson's rule.

So Simpson's rule is saved - though I would hope that those doing numerical integration would go further and use Romberg's method which can be extrapolated to any required degree of accuracy.

[Tony O'Hagan acknowledges the error, and retracts his criticism - Ed.]

An Instruction Sheet for Introducing APL to Lay Audiences (Teachers)

by Zdenek V. Jizba

Computers as a Tool for Thinking

The purpose of this DEMO is threefold:

3. To show a proposed sample session in class
2. To program in front of you EVERY step of the session
1. To ask you, the experts, to tell us how to improve this session

Because of the ease of programming, I hope that you will consider learning how to use this instructional language. (To get very proficient at it does take more

time and effort.) At the high school level, I would even suggest some of the students learn it.

The screen of this PC can be used in much the same way as you would a blackboard:

```

          5+4
9
      'TEXT'
TEXT

```

You write numeric expressions in the normal way, but must place text in quotes. The computer "EVALUATES" your expression, and prints the RESULT on the next line.

If you do not want to see the RESULT, you can divert it to a VARIABLE:

```
HIM+ 'GEORGE BUSH'
```

Now we are ready to start "programming". Actually the process is called DEFINING A FUNCTION.

We begin with *PLUS*, the simplest function of all

```
PLUS:  $\alpha + \omega$ 
```

The symbol α means "use the values to the left of" +, and the symbol ω means use everything to the right. Now let us try it!

```

          4 PLUS 3
7
          2 PLUS 1 2 3
3 4 5
          1 2 3 PLUS 5
6 7 8

```

What do we mean when we say: α means "use values to the left of"?

```

          10*1 2 3 PLUS 5
60 70 80

```

α STOPS when it finds something other than a valid value.

Now we can define *MINUS*:

```

MINUS: α-ω
4 MINUS 3
1
4 PLUS 2 MINUS 5
1

```

We could go on and define *TIMES* and *DIVIDE*, but let us now look at text:

```

HIM
GEORGE BUSH
ME←'PAUL GARCIA'
ME
PAUL GARCIA

```

We will define the function *REVERSE*:

```

REVERSE: φω
REVERSE ME
AICRAG LUAP

```

Can we use *REVERSE* with numbers? We sure can!

```

REVERSE 3 4 5
5 4 3
NOS+1 2 3 4 5
NOS PLUS REVERSE NOS
6 6 6 6 6

```

But let us return to "text" processing:

```

VOWELS: (ωε'AEIOUY')/ω
VOWELS ME
AUAIA

```

You may want to use a *DUMMY* function like *IN*, to do nothing:

```

IN: ω
VOWELS IN ME
AUAIA

```

Let us define some more *FUNCTIONS*:

```

TABLE: αρω
FLIP: φω

```

Now let us see what we can do with these:

```

WORDS+3 3 TABLE 'GEMAREBAN'
WORDS
GEM ARE BAN
REVERSE WORDS
MEG ERA NAB
FLIP WORDS
GAB ERA MEN
REVERSE FLIP WORDS
BAG ARE NEM

```

You could try to have students find words that mean something backward and forward. They can change any letter in table *WORDS* with the following expression:

```
WORDS[3;3]+'T'
```

(I-APL unfortunately does not support an easier way to do that.)

Now let us go back to numbers:

```

COUNT: ω
ODDS: (2×COUNT ω)-1
SUM: +/ω

```

Note that in defining *ODDS*, we used the previously defined *COUNT*. You can even define a function that uses itself! But now, let us see what these functions will do:

```

COUNT 10
1 2 3 4 5 6 7 8 9 10
ODDS 10
1 3 5 7 9 11 13 15 17 19
SUM COUNT 10
55
SUM ODDS 9
81
SUM ODDS 8
64

```

As you can see we could go on studying the functions we already defined, or define even more functions. You could write your own lessons, if you would be willing to learn about those special symbols and how they work.

Curve-fitting in APL

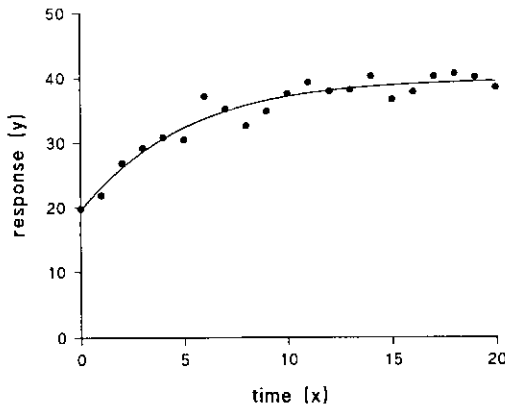
by D R Appleton (University of Newcastle upon Tyne)

Abstract

Non-linear curve-fitting of well behaved functions is easily performed in APL. Because of APL's notation it is also simple to fit functions which might otherwise be trickier.

Introduction: fitting a simple curve

Example 1.



Suppose data are available, as shown above, for a process which can be expressed in the form:

$$Y = y_{\infty} - (y_{\infty} - y_0) e^{-cx}$$

The response y might be a concentration, or the logarithm of a concentration, of some substance in blood, air or water; and we shall take time x to be measured in minutes. In APL this function and its derivatives with respect to its parameters can be written as follows, using a , b and c to represent y_{∞} , y_0 and α respectively:

```
fn ← 'a - (a-b) × * -c×x'
fa ← '1 - * -c×x'
fb ← '* -c×x'
fc ← 'x × (a-b) × * -c×x'
```

From initial estimates the parameters may be estimated by successive repetition of the line:

$$\square + (a \ b \ c) + (y - \text{fn}) \boxtimes (\text{fa}), (\text{fb}) \text{ AND } (\text{fc})$$

assuming the data values are in x and y and that M AND N is, for example,

$$M, [2-0.5 \times 1 = \rho \rho N] N$$

If the initial estimates are reasonable, the iterative process will converge quickly; the fitted curve is also shown in Figure 1. As it is inefficient to evaluate the exponential function so often, we could write:

$$\begin{aligned} \text{fn} &+ 'a-g \times a-b' \\ \text{fa} &+ '1-g' \text{fb} + 'g' \\ \text{fc} &+ 'x \times g \times a-b' \end{aligned}$$

and repeat the two lines :

$$\begin{aligned} g &+ * -c \times x \\ \square &+ (a \ b \ c) + (y - \text{fn}) \boxtimes (\text{fa}), (\text{fb}) \text{ AND } (\text{fc}) \end{aligned}$$

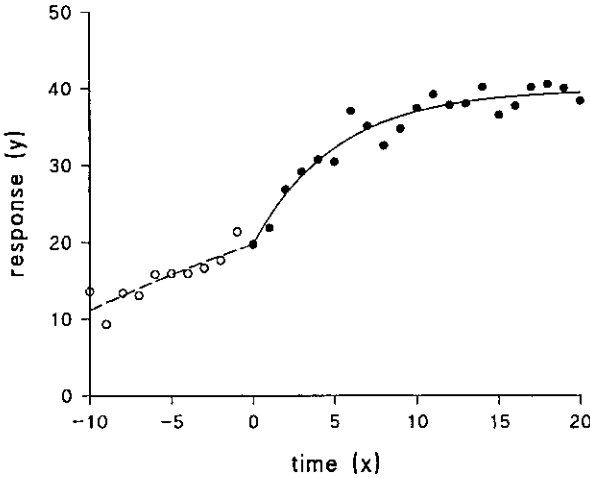
A Trivial Extension

Now suppose that we have further information, namely that the process has been operating at a constant value until time zero, and we have data not only at that time but also at times from -10 minutes to -1 at 1-minute intervals. This, of course, is easily incorporated into our curve-fitting by supposing all the readings to have been made at time 0.

A Further Extension

Example 2. But what if our information about the history of the process is less trivial? Suppose in fact that during the previous 10 minutes the process was describable by the same equation, but with a replaced by α another rate parameter β , and the data appear as shown in Figure 2.

Figure 2



We do not wish to fit the two processes separately, and in some way combine our estimates of the common parameters; we wish to fit the two equations as a single curve, and here is where APL's notation makes life so easy. If we let d play the part of β and write

$$g + * -x \times (c \times x > 0) + (d \times x \leq 0)$$

then we can leave fn , fa and fb as they are, and write:

$$fc + 'x \times (x > 0) \times g \times a - b'$$

$$fd + 'x \times (x \leq 0) \times g \times a - b'$$

The iterative procedure is the same as before, except that there are 4 parameters.

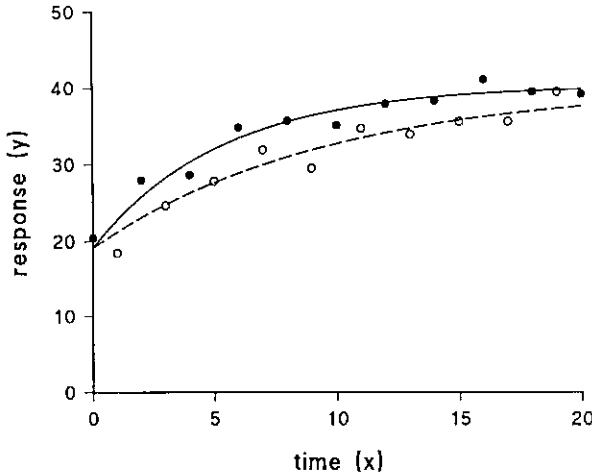
Another Extension

Example 3. As well as being able to handle situations where the function to be fitted changes (at a known value of x), the same methodology can cover simultaneous fitting of two or more curves with some parameters in common. Suppose now that we have observations on two processes

$$y_i = \gamma_\infty - (\gamma_\infty - \gamma_0) e^{-\alpha_i x} \quad (i=1,2)$$

and that observations are made every minute, alternately on y_1 and y_2 , from time 0 to 20 minutes, as shown in Figure 3.

Figure 3



If we now take as the function g the expression

$$* -x \times (c \times 2|x+1) + (d \times 2|x)$$

then the function to be fitted and its derivatives with respect to the parameters are given by:

$$\begin{aligned} fn &+ 'a - g \times a-b' \\ fa &+ '1-g' \quad fb + 'g' \\ fc &+ 'x \times (2|x+1) \times g \times a-b' \\ fd &+ 'x \times (2|x) \times g \times a-b' \end{aligned}$$

Once again the iterations proceed smoothly, and the fitted curves are plotted in Figure 3 with the data.

Naturally, it is sometimes harder to express the function as succinctly, but if all the observations of one process are arranged to precede those of the other in the x and y vectors, it will not be difficult to come up with a suitable expression.

Precision of the Parameter Estimates

Unfortunately, while the use of the dyadic domino function greatly simplifies the process of estimating the parameters, it also bypasses the calculation of the

information matrix and therefore does not allow estimation of their standard errors. Once the iterations have converged the following lines produce them (for a 4 parameter curve):

```
resids ← y - ffn
derivs ← (fpa),(fpb),(fpc) AND (fpd)
varcov ← (B(∂derivs) +.× derivs) × (+/ resids×resids) ÷ -4ρx
sterrs ← (1 1 ∂ varcov) × 0.5
```

If the line estimating the parameters iteratively is extended to include the assignments to resids and derivs, only the last two lines are required.

Functions of the Parameter Estimates

It is often possible to derive adequate estimates of the precision of functions of the parameters. Such functions might be just the sum or difference of 2 parameters, or the expression for the area under the curve, or perhaps its value or derivative at some point. The standard errors of such derived functions depend on the variance-covariance matrix of the parameter estimates (evaluated above as varcov) and the derivatives of the functions with respect to the parameters.

Suppose we wish to estimate the area between the two curves in example 3. This is given by $(y_{\infty} - y_0)(1/\alpha_2 - 1/\alpha_1)$ or in APL terms $(a-b) \times +/\div d, c$ and we could evaluate the derivatives algebraically if we wished. However, it may be instructive to calculate them numerically. Using function Δ DERIVS:

```
∇ Z←par ΔDERIVS fn
[1]  fpar, '+', par, '×1.0001'
[2]  Z←f fn
[3]  fpar, '+', par, '÷1.0001'
[4]  Z←10000 × (Z - ffn) ÷ fpar
∇
```

we can construct function DERIVS:

```
∇ Z←pars DERIVS fn
[1]  ((∂pars)←pars) ΔDERIVS "cfn
∇
```

which will produce numerical approximations to the required values. The value and standard error of the required area are given by:

```
farea ← '(a-b)×+/\div d,c'
der ← 'abcd' DERIVS area
(+/\varcov × der *.× der) × 0.5
```

Discussion

Curve-fitting is only a useful technique when there is a good reason for it. Too often it is just a way of smoothing data without thereby gaining any useful insight into the underlying process. However, if the fitting of a particular equation can be justified theoretically, and its parameters or functions of them can be interpreted, it may be valuable.

This article has shown that an APL program which can fit simple equations may also without difficulty be used for more interesting cases. A good program will, of course, do much more than has been attempted in this short exposition. It will certainly cope with any number of parameters up to a specified limit; some indication of one way to handle this is given in *DERIVS*. It will also, most importantly, check that the user has not made a mistake in his expressions for the derivatives of the function; this need was another reason for including *DERIVS*. Other checks and decisions, certainly on the iterative process itself to make sure that the parameter estimates do not exhibit large changes, and to decide how convergence is to be assessed, are required; reports on possible outliers are also advisable, and the output must be clearly labelled and expressed to a number of decimal places which reflects the convergence criterion. A library of standard curves for which all the necessary expressions exist would be a useful adjunct.

Perhaps someone will write such a program for the APL Statistical Library.

APL print on some Star printers with I-APL

by Anthony Camacho and Tom Goodman

This issue: how to print from I-APL/PC using graphics or download characters on a Star LC-10. In the next Education Vector: how to print from I-APL/BBC using graphics on a Star Radix-10.

First the LC-10

1. A complete solution is provided by the graphics output option. Start I-APL with the /G option (among others maybe) Switch on printing with `⎕HC←1` Demonstrate it by outputting a character table with `(32ρ0 1)\14 16ρ32+⎕AV` Here is the result:

```
(32ρ0 1)\14 16ρ32+⎕AV
! " # $ % & ' ( ) * + , - . /
0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O
P Q R S T U V W X Y Z [ \ ] ^ _
` a b c d e f g h i j k l m n o
p q r s t u v w x y z { | } ~ ♂
Ⓢ Ⓣ Ⓤ Ⓥ Ⓦ Ⓧ Ⓨ Ⓩ ⓐ ⓑ ⓓ ⓔ ⓕ ⓖ ⓗ ⓘ ⓙ ⓚ ⓛ ⓜ ⓝ ⓞ ⓟ ⓠ ⓡ ⓢ ⓣ ⓤ ⓥ ⓦ ⓧ ⓨ ⓩ ⓪ ⓫ ⓬ ⓭ ⓮ ⓯ ⓰ ⓱ ⓲ ⓳ ⓴ ⓵ ⓶ ⓷ ⓸ ⓹ ⓺ ⓻ ⓼ ⓽ ⓾ ⓿
Ⓚ Ⓛ Ⓜ Ⓨ Ⓩ ⓐ ⓑ ⓓ ⓔ ⓕ ⓖ ⓗ ⓘ ⓙ ⓚ ⓛ ⓜ ⓝ ⓞ ⓟ ⓠ ⓡ ⓢ ⓣ ⓤ ⓥ ⓦ ⓧ ⓨ ⓩ ⓪ ⓫ ⓬ ⓭ ⓮ ⓯ ⓰ ⓱ ⓲ ⓳ ⓴ ⓵ ⓶ ⓷ ⓸ ⓹ ⓺ ⓻ ⓼ ⓽ ⓾ ⓿
Ⓚ Ⓛ Ⓜ Ⓨ Ⓩ ⓐ ⓑ ⓓ ⓔ ⓕ ⓖ ⓗ ⓘ ⓙ ⓚ ⓛ ⓜ ⓝ ⓞ ⓟ ⓠ ⓡ ⓢ ⓣ ⓤ ⓥ ⓦ ⓧ ⓨ ⓩ ⓪ ⓫ ⓬ ⓭ ⓮ ⓯ ⓰ ⓱ ⓲ ⓳ ⓴ ⓵ ⓶ ⓷ ⓸ ⓹ ⓺ ⓻ ⓼ ⓽ ⓾ ⓿
Ⓚ Ⓛ Ⓜ Ⓨ Ⓩ ⓐ ⓑ ⓓ ⓔ ⓕ ⓖ ⓗ ⓘ ⓙ ⓚ ⓛ ⓜ ⓝ ⓞ ⓟ ⓠ ⓡ ⓢ ⓣ ⓤ ⓥ ⓦ ⓧ ⓨ ⓩ ⓪ ⓫ ⓬ ⓭ ⓮ ⓯ ⓰ ⓱ ⓲ ⓳ ⓴ ⓵ ⓶ ⓷ ⓸ ⓹ ⓺ ⓻ ⓼ ⓽ ⓾ ⓿
```

2. Only a limited solution is possible using downloads because the Star does not allow characters coded from 128 to 159 (inclusive) or 127 or 255 to be downloaded to.

The experiments I did with the Star LC-10 were done with all DIP switches ON except for 2-1 which must be OFF to allow downloads to be used. The printer must be in draft mode before downloading (see manual, page 82). It is possible that other DIP switch settings may work: I haven't tried them and they may cause complications (e.g. with the coding of the pound sign).

3. To use downloads gives the advantage that more than one size of print may be used (but only in draft mode) and that printing is faster. For I-APL to use the download set it must be started with the /E option.
4. I-APL provides a workspace *EPSON* with which to download characters. When this is used the underlining of the last two alphabets is not reproduced (see the I-APL manual or the graphics printout above). The last two alphabets are copied from the built-in character set using the command 'Copy standard characters from ROM into RAM' which is explained on page 82 of the Star manual.
5. The workspace *EPSON* contains a variable *FONT* which consists of the numeric values of the characters required to be sent to the printer to download characters for APL. In I-APL the command `2 □TX FONT` sends these values as characters to the printer without 'interpretation'. (Interpretation means that when output is sent to the printer some characters may be used to control the process and others may be converted from an internal code to something different on output. For example on daisy-wheel APL printers the lower case letters are printed as capital letters underlined and so each is sent as 'letter, backspace, underline').
6. The *FONT* variable is very easy to analyse or amend. It contains 839 codes. Here is a rough analysis of it. It is an APL vector (APL allows 'strings' of numbers) and below I have inserted new lines and comments for clarity.

```
27 64                a Resets the printer
27 58 0 0 0          a Copies ROM characters to RAM
27 38 0 128 150      a Says download of chars 128-150 follows
```

a Each downloaded character is sent as 12 bytes; a mode byte is followed by 11 dot image bytes as explained in the Star LC-10 manual on pages 82 and 91.

```
140 80 136 20 130 81 34 20 40 80 0 0
```

a The first downloaded character (∇)

a After 23 sets of 12:

```
27 38 0 158 192      a Says download of chars 158-192 follows
```

a Later downloads are for chars 219-224 and 251-254

7. So `2 □TX FONT` defines all the characters needed by I-APL. It seems to define the characters permitted by the Star printer even though some of the characters specified fall outside the permitted range. After `2 □TX FONT` the following APL command will print out a table replacing the control characters with asterisks (they perform actions like form feed which waste paper).

```
(32ρ0 1)\16 16ρ(32ρ'+'),(96+32+□AV),(32ρ'+'),160+□AV
```

You will notice that the delete character (at the end of the eighth row) and the last character show as blanks. The characters on the two rows after the eighth would have been defined as shown in the graphics output. The only characters in this set that might be needed are the pound sign and the omega. Only the omega is essential for APL.

8. The *FONT* tries to define the omega character in position 158 (hexadecimal 9E). This was done because difficulty was found in downloading to position 255 and so the internal code 255 is converted 'interpreted' on output to 158. Unfortunately 158 is not definable on the Star. To provide an omega printable on the Star LC-10 it is therefore necessary to choose another character code to be defined as omega and to change the I-APL interpretation so that instead of changing 255 to 158 it changes it instead to the code we choose. I suggest that the lower case underlined alphabet is very rarely used and that we could afford to put the omega where that alphabet has its w (code 247 or hexadecimal F7).
9. To create a *STAR* workspace with a download *FONT* which will do this follow the steps below:

```
)LOAD EPSON
)WSID STAR
FONT←FONT,27 38 0 247 247 140 28 34 0 2 12 2 0 34 28 0 0
```

A It is good practice to amend the comments so in the QLX function delete 'EPSON FX' and insert 'STAR LC-10'

```
)SAVE
)OFF
```

Now the file *IAPL.EXE* must be amended by changing the character that 255 (FF hexadecimal) is converted to from 9E to F7. The instructions below assume that you are working on I-APL/PC version 1.1 (as shown at sign on).

Everyone gets a copy of *DEBUG* with DOS so I give the sequence to follow using *DEBUG* below. As *DEBUG* refuses to write back to disk a file of type *EXE* or *HEX* you must make a copy of the *IAPL.EXE* file with a different extension. Below I have not included disk drive letters or shown the DOS prompt which depends on the configuration of your machine. To do all this on a single drive machine you should make a disk with a copy of *IAPL.EXE* plus *debug.com* and *fc.com* and put it in the *A>* drive.

```
copy iapl.exe iapl.tmp Now we can amend iapl.tmp with debug
debug iapl.tmp Load debug with IAPL.TMP. Debug prompts with '-'
the minus sign
```

```
-S 0 FFFF 81 7E FE FF 00 75 C4 C7 46 FE 9E
```

```
6FFA:0ACD The S 0 FFFF command searches from 0 to FFFF for
the string of hexadecimal bytes that follow. The
```

position found is 6FFA:0ACD. The first four hexadecimal characters may vary depending on where debug loads itself in memory

```
-D0A80      D<loc> displays the block beginning <loc>. Choose the
           hex value ending 80 or 00 just less than the location
           found; it will contain the 9E we want (end of search
           string 81 7E FE FF 00 75 C4 C7 46 FE 9E).
           This is the instruction to replace FF with 9E.

-E0AD7      Enter E 0AD7 (the location of the 9E)<return>;
           debug gives location and contents & waits for input
           Enter the new value F7 after the dot and <return>
2AF0:08D7 9E.F7
-D0A80      Check that the change is made in memory
-W          At the minus W writes IAPL.TMP back to disk
           and reports the number of bytes written
Writing 1AE9A bytes
-Q          Quit debug
fc iapl.exe iapl.tmp  Use file compare (fc) to check the only difference is
           the one we want

000009D7: 9E F7      And it is. (Incidentally if I fc iapl.tmp iapl.exe
           fc found no difference; use something else).

del iapl.exe      Delete the original program (have you kept a copy?)
ren iapl.tmp iapl.exe  Rename the amended file
-S 0 FFFF 81 7E FE FF 00 75 C4 C7 46 FE 9E
6FFA:0ACD
-D0A80
6FFA:0A80 FE 3D 20 00 74 67 8B 01-00 50 8B D9 04 50 8B 04  .= .tg...P...P..
6FFA:0A90 00 50 8B 5E 48 83 C4 06-B8 01 00 50 8D 46 FE EB  .P.^H.....P.F..
6FFA:0AA0 41 80 3E E7 03 47 75 0B-FF 76 FE EB 68 FE 83 C4  A.>..Gu..v..h...
6FFA:0AB0 02 EB 3A 80 3E E7 03 45-75 21 B1 7E FE 97 00 7C  .t..>..Eu!..~...f
6FFA:0AC0 0C 81 7E FE 9E 00 7F 05-C7 46 FE 9F 00 81 7E FE  ..~.....F.....~
6FFA:0AD0 FF 00 75 C4 C7 46 FE 9E-00 EB 8D 8B 01 00 50 8D  ..u..F.....P..
6FFA:0AE0 46 04 50 8B 04 00 50 EB-09 48 83 C4 06 5E 8B E5  F.P...P..H...^..
6FFA:0AF0 5D C3 55 8B EC 81 EC 08-02 57 56 C7 46 F8 00 00  1.U.....WV.F...
-E0AD7      0AD7
6FFA:0AD7 9E.F7
-D0A80
6FFA:0A80 FE 3D 20 00 74 67 8B 01-00 50 8B D9 04 50 8B 04  .= .tg...P...P..
6FFA:0A90 00 50 8B 5E 48 83 C4 06-B8 01 00 50 8D 46 FE EB  .P.^H.....P.F..
6FFA:0AA0 41 80 3E E7 03 47 75 0B-FF 76 FE EB 68 FE 83 C4  A.>..Gu..v..h...
6FFA:0AB0 02 EB 3A 80 3E E7 03 45-75 21 B1 7E FE 97 00 7C  .t..>..Eu!..~...f
6FFA:0AC0 0C 81 7E FE 9E 00 7F 05-C7 46 FE 9F 00 81 7E FE  ..~.....F.....~
6FFA:0AD0 FF 00 75 C4 C7 46 FE 9E-00 EB 8D 8B 01 00 50 8D  ..u..F.....P..
6FFA:0AE0 46 04 50 8B 04 00 50 EB-09 48 83 C4 06 5E 8B E5  F.P...P..H...^..
6FFA:0AF0 5D C3 55 8B EC 81 EC 08-02 57 56 C7 46 F8 00 00  1.U.....WV.F...
-W
Writing 1AE9A bytes
-Q
```

Now you can run I-APL with the /E option,)LOAD STAR, 2 □TX FONT and then proceed to load and use other workspaces which will now print everything correctly (except the pound sign).

Since writing this I have heard from WH Davies to say that his printer stops at the DEL character (ASCII 127) so it won't print the complete □AV unless you remove this character. He also points out that the download font works if the printer is set to PICA but is imperfect if it is set to ELITE.

THE RANDOM VECTOR

The Newsletter of the APL Statistics Library

Editor David Eastwood

January 1992

Contents

Editorial	David Eastwood	34
Gregynog 91	Anthony Camacho	35
ASL standards	David Eastwood	46
Differences in second generation APLs	Maurice Jordan	52

Editorial

Most of this issue of Random Vector is devoted to the third ASL conference held at the University of Wales Conference Centre at Gregynog in October 1991. Anthony Camacho was kind enough to produce detailed notes of the proceedings and these follow this short introduction.

The Conference was useful in that we looked back at the first three years of the project and drew some conclusions from our activities. We were also able to consider some new areas where the ASL project might consider producing software. One of the conclusions drawn at the conference was that the work of the authors of the next ASL volumes would be made somewhat easier if they were given guidelines for the design and production of APL code, and were also given some warnings about the differences that exist between the various dialects of APL likely to be used in the ASL project.

I include, in this issue of Random Vector, some thoughts on APL style guidelines that have emerged from the ASL project, and also a paper produced by Maurice Jordan which covers language differences between the various second generation APLs. The latter two documents are intended to be 'working papers' for the ASL project, and so we would welcome comments and additions.

Following the ASL conference, the ASL Management Committee is now:

Chairman: Jake Ansell, Business Studies, University of Edinburgh,
William Robertson Building, 50 George Square,
Edinburgh EH8 9JY
031 650 3806

Deputy Chairman: Alan Sykes, European Business Management School,
University of Wales, Swansea, Singleton Park,
Swansea, SA2 8PP
0792 295296

BAA Representative: David Eastwood, MicroAPL Ltd, South Bank Technopark,
90 London Road, London SE1 6LN
071 922 8866

BAA Representative: John Searle, 4 Hawks Mews, Greenwich,
London, SE10 8RA
081 858 6811 (home)

The 3rd APL Statistics Library Conference Held at Gregynog 30 September to 2 October 1991

reported by Anthony Camacho

Twenty people assembled in the second seminar room on Monday 30 September.

Opening Session

Alan Sykes opened the conference by going through a frank 'state of the project' letter from Tony O'Hagan (this was printed in full in the last issue of Vector). Alan Sykes then led a discussion on the current status of ASL, and the plans for future distribution. The statistics textbook to go with ASL is being worked on by Alan Mayer and he has done three chapters and two half chapters so far.

In a discussion on how ASL ought to be advertised and distributed, Alan Sykes wished to see it given away free to APL Association Members. Norman Thomson's view was that any income required should be obtained by publication of books rather than delivery of software. Nobody disagreed with Alan Sykes when he urged that we must release the work soon.

Testing ASL across Interpreters

In the first of the presentations, Maurice Jordan reviewed his work on the automatic testing of APL software. (for details, see his paper 'CATS: Computer Aided Testing of APL Software' given at APL91). Maurice said that his test bed had been improved by its use on the bottom shelf. He showed us a page (of three) with results from a function which contained only one executed line. The function was specified to produce a scalar result from two numeric non-negative vectors of matching shapes. As well as testing the function with defined pre- and post-conditions, CATS explores mutations on the test data to see if edge conditions will show errors or additional functionality.

Maurice also discussed areas of arithmetical difference between APL interpreters, for example the ? random number generator. One cause of such differences is, of course, the variation between the C compilers for the target machines. Differences between implementations are many; all use different methods for error trapping and, surprisingly, all flip to exponential notation at slightly different points. Norman Thomson felt that the ASL project ought to conserve effort and test and prove algorithms rather than implementations.

Utility Functions for ASL

David Eastwood said the utility software that would seem to be needed for ASL as it now exists doesn't match our original ideas; probably because our original ideas were wrong. We had expected to have to organise ways to bring functions to do what we wanted into the current workspace because the volume of ASL functions would be too great to)COPY complete workspaces into the current workspace, and also because there would be several variants of most ASL functions.

At present the three volumes, Basic Statistics, Basic Graphics and Regression have not led to severe problems through the space they occupy [but Karen Hurrell later said she had problems with both workspaces and symbol tables]. We need to think again about distribution systems and to know more about the target user so that we can suit the distribution system and other utilities to his needs. David thinks we diverged too early onto several platforms. We ought to get each volume working and fully tested on one platform and only then port and test the porting to other platforms.

There are problems porting between platforms because the workspace interchange standard specified by ISO is not fully implemented by any manufacturer. The IBM)IN and)OUT are useful for transfer between various versions of APL2. APL*PLUS/PC provides *IBMINOUT* and APL*PLUS II has SLT (Source Level Transfer). Dyalog can import from APL2/PC and APL*PLUS/PC. Peter Lewis said he has software to go from APL*PLUS/PC to APL2/PC. Anthony Camacho said he had received some software from Dick Holt for workspace transfer between various implementations including Sharp APL/PC and I-APL, but had not yet investigated it.

David Eastwood suggested we collect WS transfer software but reiterated that there is no point in converting before completion of the testing and he listed the steps in testing and porting to emphasise that it is not a trivial task. Peter Fisk said that some other statistics programs had sets of data with the correct results for retesting.

David said that although we now saw fewer problems with selecting functions from ASL for use, we still need library management utilities. Users will benefit from simple ways to identify alternative versions of functions and when a non-default version is selected then they will welcome a utility to amend all the calls so that they use the selected version instead of the default. We also need development tools and David recommended we look at public domain sources, such as the Toronto SigAPL Toolkit which contains *lister*, *cross-referencer*, *renamer*, *comment extractor* and *checker of localisations* - to name but a few. David pointed out that the benefit of using this sort of code is that it is written in ISO APL and can be used in most dialects.

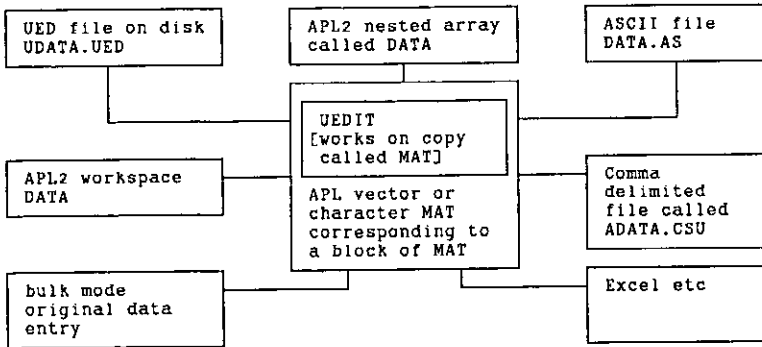
Bottom Shelf Specification

Norman Thomson distributed a revised specification of 63 pages all written using mainframe APL2. Norman feels, although some see a theoretical advantage in staying with ISO APL, that the advantages of APL2 are overwhelming; that it is very well in tune with the things that we want to do in ASL; that we aren't exploiting it properly. He then went through some of the significant changes introduced in the new specification

document. Norman wants one version to be the master specification and to write it in mainframe APL2. Norman discussed in some detail his suggestions for argument naming conventions; these are fully documented in the draft ASL Standards, later in this Random Vector.

UEDIT

In his introduction, Alan Sykes remarked that we were very lucky to be able to welcome Peter Lewis of the Department of OR in the Naval Postgraduate School at Monterey, California. Peter put up a slide of the structure of UEDIT, which is a Universal Editor and spreadsheet with many statistical functions built in.



Peter told us that UEDIT originated as 70 pages of APL code written by one of his students and has been improved over the years. He then demonstrated it using a spreadsheet of 177 rows and 11 columns. The codes for manufacturer name and description of deliverable item were converted into descriptive character vectors.

The spreadsheet offers many menus and actions, including most of the standard statistical analyses. At any time the user may enter any APL command. The spreadsheet is held as a nested matrix *MAT* and also in its original form as *DATA*, so *MAT* can be amended, used for calculations and printed without affecting the original data, which is still available for use in APL expressions. Amendment is simply done by moving the cursor to the field and typing over its previous contents. Entries are validated so character entries in numeric fields are trapped and give rise to a warning. The demonstration convinced us that UEDIT served all the needs of a spreadsheet and general matrix editor with a very broad choice of statistical analyses and particularly powerful methods for importing and exporting data from and to other systems. UEDIT is written in APL232/PC. Asked about speed Peter said if it doesn't go fast enough or hold a large enough spreadsheet "Buy a faster, larger computer.;" he had found a 16Mbyte 80486 more than adequate for anything he had tried.

A Time Series Volume for ASL

Alan Hawkes listed

1. some material already available; NAG, Genstat, S, IP Sharp workspaces, StatGraphics, Best (a Norwegian package written in MIPS APL), BATS (Bayesian Analysis of Time Series - copyright Neville Davis, Nottingham) and
2. some sources of help and further material; A G Hawkes, Clive Taylor, FinnAPL, Andy Pole (now at Duke University), plus maybe the authors of the above packages.

Peter Lewis said that S is unusable on large data sets: APL is much faster. Alan suggested the following contents for the Time Series volume, not necessarily in this order:

- Univariate TS, containing
 - Non-parametric Estimation - ACF, PACF
 - Modelling - ARIMA: Autoregressive Integrated Moving Average, Theoretical Functions, Fitting, Testing, Forecasting
- Simulation
 - Seasonal Models - Holt Winters, X-11
 - Utilities - Fast Fourier Transforms, Filtering
 - Advanced - Bootstrapping, Cross-validation
- Multivariate TS
- State space models
- Non-linear models
- Non-stationary TS
- Econometrics
- SASE (point processes) [Statistical Analysis of Series of Events]

Peter Lewis has already written up SASE so perhaps it should be chapter 2. Alan Hawkes then offered some thoughts on the best way to get the job done. He suggested the syntax should be designed and prepared as functions containing nothing but comments, and when agreed then coding could begin. Peter Lewis said that SASE began as Fortran and was rewritten in APL. The graphics from StatGraphics has been separated out and sold as APLLOT, but his students dislike it and prefer GraphStat (the APL front end to GDDM). Norman Thomson likes the principle of recording a graph as two elements: the graph specification and the data.

Jake Ansell said that John Chambers emphasised linking to non-APL code, but that he was not sure of its desirability. Norman Thomson said that NAG would always be bigger than ASL and that if ASL could link to NAG then we could offer the advantages of APL on data manipulation as an addition to those of NAG. Norman has obtained the professional Fortran version of NAG and is trying to link APL2/PC to it but has found that it would have been easier and more valuable if he had obtained the newer Fortran II version. There was a short discussion of sources of FFT algorithms and code. Peter Fisk wondered whether NAG could be an agent for the distribution of ASL and

Norman Thomson thought it could. Jake Ansell didn't want to make NAG necessary to ASL users. Peter Lewis suggested that we collect utilities for linking and gave his opinion that the best source for Time Series algorithms was a book called Time Series Analysis published by Brooks Cole but the name of the author (who comes from Texas) he was temporarily unable to recall.

Alan Sykes said that we need three or four people to work on sections of chapter one: when the specification is agreed then perhaps some of the coding could be done as student exercises. It was suggested that John Pemberton could be asked to help.

Postscript: A third year project is under way at Swansea, with supervision by A G Hawkes and J Pemberton.

A Non-linear Regression Modelling Volume

Peter Hingley circulated five pages of functions and test results and showed an example which could not be modelled as a polynomial (it plotted the response against the concentration of a drug dose). Because each point plotted was part of the data for plotting the next point the algorithms were inherently looping and this made them slow. With TryAPL2 he had processed 500 data points in 45 minutes, but he wanted to be able to handle 5000 or even 20,000 data points. He concluded by asking whether this would be an appropriate item for the regression shelf; those present clearly thought so.

Non-Parametric Methods

Karen Hurrell discussed topics in non-parametric statistics including density estimation and smoothing (non-parametric regression). Tony O'Hagan later expressed interest in a volume on this topic.

I-APL/Mac and ASL

Ian Clark began by asking 'Why on earth use I-APL as it is slow and of limited capacity?' He answered the question by saying I-APL is free, runs on many machines and offers a suitable basis for classwork, which, being ISO-conforming, could become a textbook. Ian sees himself as a Human Factors consultant and gave us some insight into how he had gone about analysing the tasks involved in producing workspaces on I-APL/Mac and then designed the interface to make common tasks easier. Doing a task analysis is often long-winded and tedious because, to spot the interesting places where good human factors design could make a significant difference to ease of use, one had to do a very detailed analysis where most of the steps almost seemed too trivial to merit recording. Indeed his first examples led to some jocular comments such as that he had omitted to include the task 'fetch student'.

As an example of a task made easier he explained how a script could be typed on a word processor with embedded APL expressions. The user could display this on the screen and read it, double click on an APL expression to copy it into the entry line, edit

it there if required, and then enter it to see the effect. As the results would be displayed at the end of the session, after the end of the script and hence force the session to scroll to the end, it could easily have been tedious to get back to one's place to continue reading, but at that point Ian Clark arranged that the keypad uparrow would return you to your place and the downarrow would then scroll to highlight the next executable expression.

A Contingency Tables Volume

Alan Mayer showed us some of the work he had done in his workspace *ASLTAB*. He suggested that a volume on contingency tables should begin by including a list of 2×2 measures, a list of $R \times C$ measures and a list of ordinal data. There is also a need to extract data, perhaps from surveys. For example to obtain a count of the items (rows) in which column 18 contains a 1, an APLer could enter `+/RAWDATA[;18]=1`; he provided the function *HOWMANY* so that non-APLers could enter `RAWDATA HOWMANY 18 1` to obtain the same result. Another function is *TABULATE*. If the data is *SURVEY* and has two values in column 1, two values in column 2 and four values in column 18 then the expression:

```
SURVEY TABULATE 1 2 18 * gives the result;
24 4 2 7
15 5 2 6

16 2 1 5
10 8 2 2
```

This has two tables corresponding to the two values in column 1; each table has two rows corresponding to the two values in column 2; each table has four columns corresponding to the four values in column 18. Where the survey has four values and we are interested in picking out one to distinguish from all others, Alan has the function *DICHOTOMISE* whose arguments specify the value(s) to pick out. Another common need is to classify entries into ranges: for example in a vector of student heights, where:

```
( $\lfloor$ /HEIGHT), ( $\lceil$ /HEIGHT
148 191
```

... the heights could be classified into the ranges: up to 160, 160 to 170, 170 to 180 and over 180 by:

```
160 170 180 CATEGORIZE HEIGHT
```

The result would be the shape of *HEIGHT* and would have five values 0-4 according to the range each height fell into. Similarly the function *ORDNAL* [sic] calculates concordancy values.

A Reliability Volume

Jake Ansell pointed out that reliability overlapped Quality Assurance, Multivariate Analysis, Time Series, Medical Statistics and even Regression (through Fault Tree Analysis). Reliability includes Stochastic Processes and Data Analysis. Jake's outline of Data Analysis was like this:

1. Elementary Analysis - mostly bottom shelf items:
 - index of dispersion
 - hazard plots
 - TTT plots
2. Distributions
 - univariate
 - exponential
 - Weibull
 - extreme value, etc.
 - censoring [Norman objected that censoring was part of the data]
 - truncation
3. Covariate Analysis - much in common with the Regression volume
 - Weibull regression
 - PHM
 - ALT [Accelerated Life Testing]
4. Repairable Systems
 - trend tests
 - growth models
 - dependency analysis

Sample Surveys

Allan Prys-Williams made some ad hoc suggestions. He asked what we wanted in the way of:

1. Simple auxiliary functions
 - construction labels
 - drawing samples whose size is proportional to their probability
2. Sample drawing
 - stratified unweighted or weighted according to sub-population size
 - clusters - either one size at each level; N each of M of K
3. Costs of interviews
 - can be an auxiliary variable X with or without a known population

An Application of ASL

Maurice Jordan reported that at the end of the football season the Observer tabulated all combinations of home and away scores with the frequency of each, for the First Division and the Second Division as two tables. Maurice discovered that the scores

were a Poisson distribution. For the First Division χ^2 is 0.5; for the Second Division there were a few too many no-score draws for a good match.

Maurice said "Given a computer and a good set of statistical functions you begin to think about problems differently" and proceeded to give some fascinating examples of odd places where Poisson distributions appear and the ability to analyse them can be valuable. To win money on the pools one must forecast unlikely results. When the favourite wins, the odds are too low to make much profit.

Documentation

David Eastwood chaired a discussion on documentation. Jake began by surveying what we had so far. For the Basic Statistics each function has a corresponding *HOW* function - a basic minimum. Alongside that there is Norman's specification. For the Graphics there is Tony O'Hagan's user guide which explains how the language was drawn up and gives implementation notes. For the Regression volume Alan Sykes has produced an introduction to the statistics covered, the data required, how the functions work and some guidance on adding your own functions. There is also a nice crib sheet to say what the functions actually do. In Jake's view this would be a good standard to aim for all subsequent volumes.

Alan Hawkes suspects that the Basic Statistics, being a pot-pourri of bits and pieces, might not get more than its current fragmentary documentation. Jake Ansell said that it needs at least an introduction. Alan Prys-Williams suggested that volumes might shadow a dominant textbook and thus the textbook could become the explanatory part of the documentation. Function and variable names would have to match the usages in the book. Sometimes it would be necessary to give a result a name other than Z.

Alan Sykes felt that the WAGS and Regression documents are almost publishable. Norman Thomson suggested that we look at the problem the way a publisher would look at preparation of a series; he would employ a commissioning editor who would write a specification which all authors would be asked to follow. At the lower level he wanted to ensure that items capable of being calculated in varying ways should be made consistent. Whenever a mean is used it should be exactly the same.

There was some discussion about the use of enhanced features. On the whole the coding tried to stick to ISO APL but we had accepted some second generation phrases. The Regression volume was entirely written in ISO APL, and the Basic Statistics in its APL*PLUS version was also ISO. WAGS was only fully implemented in Dyalog and did use some advanced features.

Anthony Camacho recommended Gary Bergquist's book "Advanced Techniques and Utilities" where the functions often had alternative commented out lines for other interpreters so that to change from IBM to Sharp to STSC all one would have to do is insert and delete the comments indicated. Peter Lewis remarked that Gary was also author of the Zark APL Tutor which he recommended and which his students had found useful, instructive and amusing. It was reviewed in Vector 7.4. Maurice Jordan had once begun work on a system which would search for all primitive usage which

could be ambiguous and replaced each use with a cover function. Then by simply substituting the APL2 cover functions for the APL*PLUS cover functions, the whole workspace could be painlessly adapted to run on APL2. He thought that he had never finished the job but would try to find it to see whether it could be useful for ASL. David Eastwood thought that the least we should do was to provide notes warning all concerned about the known trouble areas when converting from one implementation to another (see the ASL draft Standards).

Norman Thomson said that he had begun to question our decision to keep the bottom shelf at a suitable level for a first year undergraduate course. Many simple and basic things would not fall into this definition and he recommended that we extend it a bit. For example weighted least squares would be a good addition. Jake disagreed; the criterion we had chosen would be understood by more people than any other we had been able to think of. He also thought that weighted least squares should be Regression.

Group Sessions

David Eastwood chaired a discussion which led to the conference breaking into two streams for the two sessions before the final plenary. One stream, with the computer and display equipment, would see a demonstration of WAGS in the first session and Maurice Jordan demonstrating partition techniques in the second (this was included because these techniques make it easy to do things in ISO APL which look as if they would be easy only with nested arrays); the other would review standards and documentation in the first session and discuss porting problems and techniques in the second.

During the discussion Norman Thomson wanted to exclude data input and output on the grounds that: (a) it isn't statistics; (b) it is very system-dependent and (c) it is something that every vendor and software house claims to be expert on. Alan Mayer felt that we shouldn't turn our backs on the possibility of taking data from Lotus or returning results to it.

Partitioning Techniques

Maurice Jordan said that Bob Smith, the implementer of STSC's NARS had worked on partitioned functions in the 1970s. These were probably the basis of partitioned enclose and they certainly enable one to operate on simple arrays in ISO APL as if they were nested. The breaks between nests are marked in a boolean variable which is otherwise all zeros, by a 1 at the beginning of each partition - in Dyalog thus:

```

P←1 0 1 0 0 1 0 0 0
P<19
1 2 3 4 5 6 7 8 9
+/"P<19
3 12 30

```

Using the partitioned technique the equivalent is:

```

P P_PLUSREDUCE 19
3 12 30

```

and the result from this is much faster than using nested arrays.

Anthony Camacho asked where these functions were available and when he was told that they were the workspace PARTFNS in WSLIB 6 in Sharp APL he realised that I-APL has the PARTFNS workspace, improved and generalised by David Ziemann and issued on every I-APL disk. This was established on Alan Mayer's portable computer and Alan thought that it would be a good subject for an article in Education Vector.

Closing Decisions and Actions

David Eastwood said the regression volume is closest to a model set of documentation that we have. Probably the Basic Statistics will be documented to a different standard and he suggested that an introduction to precede Norman Thomson's specification would be a minimum.

It was now agreed that we would not bar the use of enhancements in implementations on second generation interpreters, although we still preferred to avoid them. We do need to warn people about problems moving from one interpreter to another and he suggested that we collect these. Anthony Camacho said that Adrian Smith had spoken at APL 91 about writing in a common subset of the major interpreters; he would probably be able to provide a set of rules to follow to write functions which would run unchanged in all of them. Alan Sykes mentioned that it was necessary to take care that a nested vector containing a singleton such as in 'Alan 'M' 'Sykes' was a vector of vectors: if the singleton was a scalar the function would fail because the level of nesting would become inconsistent. A simple doubly delimited character vector provided a robust solution.

David Eastwood undertook to circulate a first draft of a paper about avoidance of problems when writing for multiple interpreters (see this issue of Random Vector). The following people agreed to be the collection points for various types of material:

Alan Mayer	Regression
David Eastwood	Utilities
Alan Hawkes	Time Series
Alan Mayer	Contingency Tables
Alan Sykes	Multivariate Statistics
Jake Ansell	Reliability
Allan Prys-Williams	Sample Surveys
Karen Hurrell	Smoothing and density estimation

Norman asked for the quality of the quantile tails specification functions to be reviewed. David Eastwood agreed to write an introduction to the specifications.

It was agreed that we had not yet solved the problem of maintaining consistency between versions of functions which might appear in more than one volume; it was

agreed that this could happen - we were not going to try to ensure that each function appeared in only a single place. It was thought a good idea to date each function and also to provide a comment indicating where it came from in the library - this would enable anyone to check that they were using the latest version.

Jake Ansell agreed to act both as Chairman of the team and Manager, with Alan Sykes as his deputy who would probably take over when/if needed. The other members of the team are Alan Mayer, David Eastwood and John Searle. Alan Sykes asked that we call the three volumes by their correct names from now on. They are ASLBSTAT for basic statistics, ASLGRAPH for the graphics (N.B. not WAGS) and ASLGREG for the general regression volume.

It was agreed that an advertisement and order form for ASLGREG should go in the next Random Vector. Alan Mayer will proof read the documentations and Alan Sykes will make 60 copies and await orders. ASLGRAPH needs the documentation changing to take out the references to Dyalog. Then the STSC version can be completed and it and the Dyalog version are almost ready. That leaves the ports to APL2 and APL.68000. It was agreed that when we issue the basic statistics we will include with it an advertisement for the other volumes on the bottom shelf. Ian Clark said he would try to put the basic statistics into I-APL/Mac.

The concept of patrons for sections of the project was discussed. It was felt that it would be helpful to approach leading APL statisticians and to ask them if they would be willing to become patrons of the ASL project; an honorary position primarily designed to formalise their support for the project.

ASL Standards

by D.S. Eastwood

Preface

At the ASL conference in October, I agreed to attempt to codify the standards that have been agreed for the ASL project. These standards are formulated as fairly loose guidelines and have been drawn up as a result of discussions during the course of the project and as a result of experiences in writing the APL code for ASL. Indeed, much of the current ASL code will need considerable reorganisation to bring it into line with these standards. We hope that the main beneficiaries of this document will be those intending to submit code to the ASL project, and, of course, others producing APL systems.

Contents

- APL language usage
- Functions and error trapping
- Naming conventions
- APL dialect differences
- ASL documentation

Appendix:

Differences in second generation APL's Maurice Jordan

References:

1. ISO 8485: 1989 *'Programming Languages - APL'* (also BS7301:1990)
2. *Basic Statistics Volume* - Norman Thomson
3. *ASLGREG - Regression and generalised linear models volume*
Version 1.1 - Alan Sykes
4. *Error trapping tutorials* - Ziemann, Eastwood, Brand
Vector 6.1 July 89
5. *Error trapping in APL2/PC* - Norman Thomson
Vector 6.4 April 90

APL Language Usage

Base dialect. It is recognised that a considerable divergence exists between the different dialects of APL, and in particular between so-called 'first generation' and 'second generation' APLs. In an attempt to minimise problems that might arise in converting between APL dialects, it is recommended that contributors to the ASL project attempt to write APL code which follows the ISO standard (Ref 1) unless some compelling reason exists to adopt a second generation feature. Simple data arrays should be used where possible.

In practical terms, the current range of APL interpreters which are used by contributors to the ASL project are:

APL2/370	Release 1
APL2/PC	Release 1
APL*PLUS/PC	Releases 6 onwards
APL*PLUS II/PC	Release 3
APL.68000	Release 7
APL.68000 Level II	Version 2
Dyalog APL	Version 6.1

A contributor to ASL should make it absolutely clear which version of APL is to be regarded as the master version for the contributed code. Note that the release number is also useful as some of the interpreters listed above are prone to add new language features when they are upgraded.

Index origin. Index origin 1 is the standard and functions which use Index Origin 0 should localise $\square IO$.

Global Variables. Global or semi-global variables are deprecated and should only be used if they do a statistical job.

System functions. \square functions should be avoided where possible as there is a high probability of differences between APLs. Even functions as innocuous as $\square NC$ can have different definitions.

Functions and Error Trapping

Specification Language. It is recognised that APL (or a nominated dialect of APL) is likely to be used as the specification language for ASL code, and it is also recognised that it is needlessly cumbersome to include, for example, full comments and error trapping in a specification document. The notes which follow apply to implementations of ASL.

Error trapping. For bottom shelf functions, errors should result only from misuse (using parameters beyond those specified), and thus error trapping should cut back the stack by one and signal the appropriate error message. For an analysis of the different schemes of error trapping used by APL interpreters, see Refs 4 and 5.

Direct definition. Is not used in ASL.

Naming Conventions (Function names). There is currently no rule that ASL functions should carry identifying prefixes. All names in ASL should be distinct and should be registered at the specification stage. It should be borne in mind that functions may have to be renamed (either to facilitate integration with user code) or to resolve name clashes. Use of *z* to construct names should therefore be avoided.

ASL function names should be meaningful without being too generic. Thus names like *PRINT* are not useful!

In many volumes groups of functions will fulfil related purposes and equally there will often be implementations of alternative algorithms. A suggested naming convention (Ref 2) is that function names should follow a general pattern:

<root> <inflection...>

where the dots indicate the possibility of multiple inflections. Implementations of alternative algorithms are given serial numbers as a further inflection. When a function calls a succession of auxiliary functions, the latter are named by applying successive Δ s to the root, or root+inflections. Thus full specification of a function name is thus:

[Δ ...]<root><inflection...><serial number>

Examples, from the Basic Statistics Volume, are:

MEAN
MEANF *mean of a frequency table*

Naming Convention (Local names). A number of alternative schemes are suggested, covering function arguments, local variables and labels. The ASL contributor to should make a clear statement of the scheme used. These schemes are listed below.

1. Results

Choose a standard name for a result (e.g. Z)

2. Arguments

- 2.1 It is possible that an argument naming convention can be directly applied from statistical theory or from a suitably acknowledged reference. Whilst attractive in terms of readability for statisticians, it is highly likely that supplementary names will have to be introduced and thus recourse will have to be made to some APL naming convention (see below).
- 2.2 Meaningful upper case names can be given to arguments (*DF* for 'degrees of freedom' in example below). Note that this does not relieve the author of the responsibility of describing the argument!
- 2.3 Arguments names can be constructed to indicate the nature of the expected argument type (Ref 2). The word will be constructed in lower case letters. For example, the following coding is used in Ref 2:

Prefix	Algebraic Type	Rank	Repeatability	Suffix
ne	b	s	v	
nd	n	v	m	
f	z	m	a	
s	(r)	a	2	
	p			
	c			
	g			

where:

ne	non-empty	s	scalar
nd	non-degenerate (non empty/scalar)	v	vector
f	frequency	m	matrix
s	shape	a	array
b	boolean	2	repeat twice

only

n	non-negative integer
z	integer
r	real (default)
q	positive reals $0 < q$
p	probability $0 < p < 1$
c	character
g	general (character or numeric)

Note that type 'fm' (frequency matrix) occurs frequently and describes a two column numeric matrix - class values and number of items in a class.

- 2.4 Other local variables may similarly be given names which follow one of the above rules, or should use the single character names

T, U, V, W

Loop counters will use the sequence

I, J, K, L

2.5 Labels should either follow the sequence

L1, L2, L3, ...

(using *EL1,...* as 'end loop' if required) or, preferably, use a meaningful name where one can be constructed.

2.6 Compound arguments will often be employed. Each component is separated by an underscore and named independently (as above). Such arguments should be explicitly decomposed:

(N A B)+N_A_B (in a second generation APL)

3. Comments should follow the standard scheme illustrated below:

```

VRES+DF CHIDEN X;Y;Z;ΔELX
[1]  A FN: RETURNS VALUES OF THE CHISQUARED PROBABILITY DENSITY
[2]  A SS: ISO/BS
[3]  A RA: VECTOR OF CHI SQUARED VALUES WHERE PROB'S ARE REQD
[4]  A LA: SCALAR INTEGER - DEG OF FREEDOM OF CHI_SQUARED DIST'N
[5]  A RE: VECTOR OF PROBABILITIES
[6]  A
[7]  ΔELX+[]ERX ERR
[8]  RES+(X*Z)×(*-0.5*X)÷(!Z+~1+0.5*DF)÷0.5*0.5*DF
[9]  +0
[10] ERR:'ARGUMENT ERROR' []ERS 8
      V 1991-10-16 0.12.36

```

with recommended comments and identifiers from:

FN: describes the purpose
 SS: shows the source - e.g. BS Basic Statistics
 RA: right argument
 LA: left argument
 RE: result
 AL: algorithm and attribution
 VE: version / date

Dialect differences

It is hoped that the attached notes covering dialect differences (see Appendix) will assist an ASL author in cultivating an awareness of areas where APL dialect differences will cause translation problems.

ASL documentation

Certain minimum levels of documentation should be supplied with ASL workspaces. This will cover the use of the workspace, interdependencies within the workspace and examples of usage. Since ASL workspaces can range from collections of largely stand-

alone functions to complete systems, the quantity of documentation in each of the categories may vary from volume to volume.

Specification document

Each ASL volume should begin with a Specification Document which outlines the scope of the volume and indicates the names and syntaxes of the functions within the volume. The specification document is used to indicate names used, naming conventions applied and data structures. Reference 2 - the Bottom Shelf specification document - is a good example of this type of document. This may be the same as the:

General documentation

This covers the overall theory and operation of the volume, with reference, if appropriate, to suitable published reference material. The ASL Regression volume (Reference 3) provides a good example of this sort of document.

Function documentation

This document will include the internal comments found within ASL functions (see above) together with:

- Example uses of the function with test data and expected results.

- Indication of the range of operation of the function (if not internally documented).

- If variants of a given function exist, a description of the variants and their differences should be included.

Workspace documentation

Each ASL workspace should be supplied with the following pieces of workspace documentation (or the means to produce them):

- Workspace listing
- Function cross reference
- Global variable listing and description
- List of functions for which an alias exists

Porting notes

Each ASL document should have some notes on porting. In the case of most workspaces this will be confined to comments regarding use of second generation APL features, and, in particular, of uses of APL features known to vary from APL system to APL system. It may also be appropriate to indicate means of testing parts of the workspace or volume as they are recoded, especially if the recoding process is not trivial.

Appendix: Differences in Second Generation APLs

by Maurice Jordan

This paper covers differences that I am aware of between a number of second generation APLs. It is not necessarily an exhaustive list, and my knowledge and available sources are far from complete. All Sharp implementations are different again, and in general are not covered.

Note the differences between APLs from the same supplier - IBM and STSC are both guilty of this. As you can see, the list is rather long, and I have not considered any System Functions at all.

In the case of code written for a given second generation APL, the probability of a function working without modification in a different implementation is only slightly higher than the probability of it working under VS APL.

Valid object names

Underbar `_` and overbar `¯` are allowed in APL2 and APL.68000 identifiers, but not as initial character. APL*PLUS/MF allows neither, APL*PLUS/PC allows `_` but not as the first character. Dyalog only allows `¯`, but this can be the initial character.

Dyalog allows uppercase, underscored, and lowercase alphas in identifiers. APL*PLUS/MF does not permit lowercase. APL2 permits uppercase and underscored with lowercase being an alternative for underscored. APL.68000 allows lowercase, although some (dumb terminal) versions will still use underscored instead.

Function headers $Z \leftarrow A \text{ } FOO \text{ } B ; C ; D ; E ; F$

Dyalog APL insists on elidable left arguments being enclosed in braces `{ }` and allows "shy" results (also denoted by being enclosed in braces). This is also true of DEC VAX APL, I believe. (DEC also allow an axis specification.) Braces are not accepted by APL*PLUS/MF, APL2, APL.68000, or any first generation APL.

APL2 allows either space or `;` as separator in the locals list. This is not true of APL.68000, Dyalog, APL*PLUS/MF, nor any first generation APL. Dyalog and APL*PLUS allow $A \leftarrow A \text{ } FOO \text{ } B$ and $B \leftarrow A \text{ } FOO \text{ } B$, as does APL2 and APL.68000.

Defined operators

The programmer may define operators with header syntax $Z \leftarrow A (F \text{ } OP \text{ } G) B$ in APL2, APL.68000 and Dyalog but not any flavour of APL*PLUS or any first generation APL. See also above remarks on braces in Dyalog.

Strand notation

Strand (or vector) notation $Z \leftarrow A \ B \ C$ ($A \ B \ C$ all variables) is allowed in APL2, APL.68000, Dyalog, APL*PLUS/MF, but not in any Sharp or first generation implementation. Strand notation involving quoted strings can differ in APL*PLUS/MF, depending on the setting of the system variable $\square CLEVEL$.

Strand notation assignment

is allowed in each of Dyalog, APL2, APL.68000 and APL*PLUS/MF. APL2 and APL.68000 syntax seems to differ from the others in that the variables being assigned to must be enclosed in parentheses.

```
A B ← A Δ B      ⍝ Dyalog, APL*PLUS
(A B) ← A Δ B   ⍝ APL2/MF, APL.68000
```

The latter causes a *SYNTAX ERROR* in other APLs. APL2/PC differs from APL2/MF, but reportedly in the curious way that it produces a different result rather than an error.

```
→ 0 Z ← EXPR
```

is illegal (rightly so in my opinion) in all but APL2 and APL.68000.

Diamond

Is supported in all APLs except APL2/MF and would be used in preference to constructs such as $\rightarrow 0 \ Z \leftarrow EXPR$ by any APLer to whom it was available.

Primitive Functions, Operators etc.

Enlist/Type ϵA . Monadic ϵ means enlist in APL2, type in Dyalog and APL*PLUS/MF. It generates a *SYNTAX ERROR* in APL*PLUS PC.

```
R ← ENLIST R; □TRAP
⍝ V REDUCE ARBITRARY APL VARIABLE R TO SIMPLE VECTOR
⍝ ENLIST (MONADIC ε) IN APL2
□TRAP ← ΔPASSBACKERRORS
→ (0 ≠ ρR ← ,R) / S
L: → (1 = |R) / 0 ◊ R ← 0 ρ R ◊ → L
S: → (1 = |R) / 0 ◊ → (0 = ρR) / L
R ← , / , "R ◊ → S
⍝ α MHJ 31MAY90 FROM MAINFRAME
```

```
A ← TYPE A
⍝ V ATTEMPT TO DEFINE DIALOG, APL*PLUS ε IN APL2/MF, APL.68000
A ← + 0 ρ C A
```

Selective specification $(F Z)+EXPR$. is allowed in APL2/MF, APL.68000 and Dyalog, but not in APL*PLUS or APL2/PC. The domain with respect to function F may differ.

Enclose $\subset A$. Same in all APL2-like second generation APLs; the basic difference is between these and Sharp implementations.

Enclose with axis $\subset [1]A$. Specific to APL2, APL.68000. Dyalog and APL*PLUS use split $\uparrow A$ or $\downarrow [1]A$.

Partitioned enclose or partition $A \subset B$. In Dyalog and APL*PLUS/MF A must be boolean. In APL2 and APL.68000 A is integer. The Dyalog $A \subset B$ is reproduced by $(+\backslash A) \subset B$ in APL2 and APL.68000. APL2 and APL.68000 allows values from B to be removed from the result, Dyalog and APL*PLUS do not, apart from the case where there are leading zeros in A . In Dyalog and APL*PLUS, for vector B ,

$$\begin{aligned}(\rho A \subset B) &= + / A \\ (\rho, / A \subset B) &= (\vee \backslash A) / B\end{aligned}$$

Pick $A \supset B$. is the same in Dyalog, APL*PLUS, APL2 and APL.68000.

Disclose $\supset B$ (APL2 APL.68000). Is mix $\uparrow B$ in Dyalog and APL*PLUS. APL*PLUS II will give a *LENGTH ERROR* on \uparrow 'FRED' 'JOE'.

First $\uparrow B$ (APL2, APL.68000). Is $\supset B$ in Dyalog and APL*PLUS.

Depth $\equiv B$. APL*PLUS/MF has two definitions, controlled by a system variable $\square CLEVEL$. The APL2 and APL.68000 definition can be represented by:

If B is a simple scalar, $\equiv B$ is 0.
Otherwise $\equiv B$ is defined recursively as $1 + \uparrow / \equiv B$.

This is the same for APL*PLUS/MF with $\square CLEVEL \neq 0$. Dyalog $| \equiv B$ is defined similarly, but $\times \equiv B$ is $\bar{1}$ if depth of nesting at any level is different. APL*PLUS/MF with $\square CLEVEL = 0$ is similar to Dyalog, but has magnitude 0 $\uparrow \bar{1} + | \equiv B$.

Axis operator. Axis operator with primitive scalar functions $A F [I] B$ does not exist in APL*PLUS or Dyalog. In Dyalog it can be simulated with a defined operator. Ravel with axis $(, [I] B)$, take with axis $(\uparrow, [I] B)$ and drop with axis $(A \downarrow [I] B)$ are present in APL2 and APL.68000 but not Dyalog or APL*PLUS/MF.

Squid \square (indexing function). Is in APL2 and APL.68000 but does not exist in Dyalog or APL*PLUS.

Bracket indexing $A [B]$. Has been extended in Dyalog and APL*PLUS, but not APL2 or APL.68000, to allow nested B (scatter-point indexing). Dyalog and APL*PLUS/MF differ on the range of nested arguments allowed for this.

Find ϵ . Does not exist in Dyalog or APL*PLUS before Release 9. For text vector arguments, it can be simulated easily in APL*PLUS using $\square SS$. The Dyalog AP SS can be used to simulate most instances.

Each operator F'' . When applied to empty data $F''E$, APL*PLUS gives a *NONCE ERROR*; Dyalog tries to return $(\rho E)\rho \subset F\epsilon \supset E$; APL2 and APL.68000 return the fill function applied to the prototype of E .

Composition operator $F \circ G$. Exists in Dyalog but not APL2, APL.68000 or APL*PLUS.

Commute operator F'' . Exists in Dyalog but not APL2, APL.68000 or APL*PLUS.

Catenate-first-axis $A \bar{\vee} B$. Exists in Dyalog, APL.68000 and APL*PLUS/PC but not APL2 or APL*PLUS/MF.

Function assignment $F \leftarrow +/$. Exists in Dyalog but not APL*PLUS, APL2 or APL.68000.

Assignment as a pseudo-operator $A \leftarrow F+B$ and $A [I] \leftarrow F+B$. Exists in Dyalog but not APL*PLUS, APL2 or APL.68000.

Set functions $A \cap B$ $A \cup B$ $A \sim B$. Exist in Dyalog. \sim (without) is in APL*PLUS PC and APL*PLUS II but not APL*PLUS m/f. Only $A \sim B$ in APL2 and APL.68000.

Unique $\cup A$. Exists in Dyalog, but not APL*PLUS, APL2 or APL.68000.

Index generator ιN . Has been extended in Dyalog to accept a non-singleton argument, producing a nested array result. This extension does not apply to APL2, APL.68000 or APL*PLUS.

Replicate and expand. Have been extended in Dyalog to take generalised left arguments. This extension does not apply to APL2, APL.68000 or APL*PLUS. In Dyalog and APL*PLUS/MF, they continue to be classed as functions rather than as operators. Thus:

$(0 \ 1) (1 \ 0) /'' \subset \iota 2$

will work. It does not work on APL*PLUS II, APL2 or APL.68000.

N-wise reduce $N \leftarrow F/B$. Is in APL2 and APL.68000 but is not present in Dyalog or APL*PLUS.

Reduction and inner-product. With non-scalar functions work differently between (Dyalog and APL*PLUS/MF) and (APL2, APL.68000 and APL*PLUS II). Non-scalar reductions on empty arguments may be different where they does not produce an error.

High level formatting. The implementations of $\square FMT$ (Dyalog, APL.68000 and APL*PLUS) and $A \nabla B$ (APL2 with text A) all are different. APL*PLUS and APL*PLUS II insert a column of blank spaces on the left of numbers - others do not ($\nabla 9 \ 1 \ \rho \ 1 \ 9$).

Default display. Differs.

The atomic vector $\square AV$. Is specific to each implementation.

Shared variables. Are specific to each implementation, and may not be present.

Error handling. The system function to signal an error is different in each implementation. APL.68000 implements APL2's $\square EA$ $\square EC$ $\square ET$ as well as its own $\square ERX$. Facilities to take alternative action on errors exist in all second generation APLs. Apart from APL2, it is possible to design utilities which take the same action in other implementations.

Complex numbers. Are available in APL2/MF but not APL2/PC, Dyalog, APL.68000 or APL*PLUS.

Roll $?A$. For some arguments e.g. $?^{-1+2*31}$, roll gives different results on different systems for the same value of $\square RL$. $?^{-2+2*31}$ seems to give the same result, at least between Dyalog and APL*PLUS/MF. The domain of roll differs between Dyalog and others. In Dyalog, it must be stored as a positive integer (up to $^{-1+2*31}$); in others it can be a "near integer" requiring floating point storage (up to $\lfloor / \iota 0$) As the roll algorithm can generate at most $^{-2+2*31}$ distinct values, it seems logical to use this value.

Note: Alan Sykes has tested $?^{-2+2*31}$ on a number of interpreters, and reports that they all gave the same results with the exception of APL*PLUS/PC. It seems that although APL*PLUS/PC uses the (implicit) standard $\square RL \leftarrow (^{-1+2*31}) \mid (7*5) \times \square RL$ algorithm for updating $\square RL$, it switches to an algorithm which uses 8 successive values of $\square RL$ to generate 1 random number when the argument to $?$ exceeds $^{-1+2*15}$.

Placement of parentheses. Different interpreters allow parentheses to be included in expressions at different points. This is true of APL*PLUS/MF versus APL*PLUS II. Usually these parentheses are only needed to deal with some nastiness in a second generation language feature.

Complex indexing. There are differences in such expressions as $A \ B \ C[3]$. I strongly recommend steering clear of these. Also, I recall problems with a benchmark involving $A[I] [J] [K]$.

Differences in arithmetic precision. Alan Sykes remarked on differences in $A \square B$ in his APL87 paper. There may be others.

Internal Storage. The number of bytes needed to store an object can vary widely between implementations. For example APL*PLUS/PC does not have boolean storage. Dyalog does not have arithmetic progression vectors. PC-based APLs have short integers. These differences can mean the difference between a successful run and an afternoon of WS FULLS.

APL Product Guide

Compiled by Alison Chatterton

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
- * APL Interpreters
- * APL Visual Display Units
- * APL character set printers
- * APL-based packages
- * APL Consultancy
- * APL Training Courses
- * Other services
- * 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 Alison Chatterton, at the address on the inside back cover.

COMPLETE APL SYSTEMS

COMPANY	PRODUCT	PRICES(£)	DETAILS
Active Workspace Ltd	AWL486	4,450	486 based 25MHz PC, 140MB Disk, 4MB RAM, VGA Colour. (inc. 1 year on site maint.)
	AWL 386	3,095+	386 based 25 & 33MHz PC, 140MB Disk, 4MB RAM, VGA Colour. (inc. 1 year on site maint.)
APL People	IBM PCs & compatibles	poa	Includes PC, mono/colour monitor, APL Interpreter, operating system software, plus optional printers, graphics boards, additional memory etc.
Dyadic	IBM RS/6000 MD320	11,738	APL POWERstation (Greyscale) 27.5 MIPS, 7.4 Mflops RISC Processor 8Mb RAM, 120Mb Disk 19" 1280x1024 Greyscale Graph Display AIX, OSF Motif, Dyalog APL (1-user)
	IBM RS/6000 MD320	13,817	APL POWERstation (Colour) 27.5 MIPS, 7.4 Mflops RISC Processor 8Mb RAM, 120Mb Disk 16" 1280x1024 Colour Graphics Display AIX, OSF Motif, Dyalog APL (1-user)
	IBM RS/6000 MD320	22,656	Advanced APL POWERstation 27.5 MIPS, 7.4 Mflops RISC Processor 16Mb RAM, 320Mb Disk, 150Mb Tape 16" 1280x1024 Colour Graphics Display AIX, OSF Motif, Dyalog APL (1-user)
	IBM RS/6000 MD520	37,114	APL POWERSystem (8-users) 27.5 MIPS, 7.4 Mflops RISC Processor 16Mb RAM, 320Mb Disk, 150Mb Tape CD-ROM Drive, 16 Ports AIX, Dyalog APL (2-8 user licence)
	IBM RS/6000 MD530	72,054	APL POWERSystem (16-users) 34.5 MIPS, 10.9 Mflops RISC Processor 32Mb RAM, 1.34Gb Disk, 2.3Gb Tape CD-ROM Drive, 16 Ports AIX, Dyalog APL (8+ user licence)
	IBM RS/6000 MD540	122,842	APL POWERSystem (32-users) 41 MIPS, 13 Mflops RISC Processor 64Mb RAM, 1.7Gb Disk, 2.3Gb Tape CD-ROM Drive, 32 Ports AIX, Dyalog APL (8+ user licence)
MicroAPL	Aurora	20,000+	Multi-user APL computer using 68020 CPU, Std. configuration 2Mb RAM, 18 RS232 ports, 68 Mb hard disc, 720K diskette
	Spectrum	7,000+	Expandable multi-user APL computer using Motorola 68000. Std. configuration 1 Mb RAM, 12/36 Mb disc, 12 ports.

APL TIMESHARING SERVICES

COMPANY	PRODUCT	PRICES(£)	DETAILS
Reuters Ltd	SHARP APL	poa	International Network application systems and public databases.
Univare	APL*PLUS	call	STSC's mainframe service

APL INTERPRETERS

COMPANY	PRODUCT	PRICES(£)	DETAILS
Active Workspace Ltd	DYALOG APL DOS 386		Dyadics PC 386 APL Interpreter
APL Software	APL*Plus/PC Release 9	535	STSC's APL for IBM PC, PC/AT and PS2. Upgrades from earlier Releases also available.
	Run-time	poa	Closed version of APL*Plus/PC which prevents user exposure to APL.
	APL*Plus II	1600	Incorporates mainframe features & performance in a version of APL for the PC
	Run-time	poa	

	Dyalog APL	1000-10,000	2nd generation APL for Unix systems
	APL2/PC	376	IBM's APL 2 for the PC.
Cooking/Drury	APL*PLUS PC Rel 10	410	STSC's full featured APL for IBM's and compatibles - Version 10 includes the Quad-NA facility to interface to non-APL software, support for Microsoft Windows and mouse devices. The User-command processor has been built in to the interpreter. Upgrades to version 10 are available from Version 9 and earlier releases.
	APL*PLUS PC Run-Time	175 for 5	Closed version of the interpreter for developers, prevents user exposure to APL
	APL*PLUS PC Developer System	950	Gives rights to distribute an unlimited number of copies of Run-Time application.
	APL*PLUS II System	1200	High powered APL interpreter for the 80386 chip. Price includes one years maintenance and free upgrades - volume discounts VERSION 3.5 NOW AVAILABLE.
	APL*PLUS II Developer System	3200	Unlimited distribution of APL*PLUS II Run-Time applications!
	APL*PLUS II for UNIX	poa	STSC's 2nd generation APL for all major Sparc and Risc Unix workstations. Version 4 for existing platforms and for the IBM RS/6000 available from January.
	APL*PLUS VMS	poa	2nd generation APL for DEC VAX computers running under VMS.
	APL*PLUS Mainframe	poa	Enhances VS APL with many high performance, high productivity features. For VM/CMS and MVS/TSO offers simple upgrade from VS APL.
Dyadic	Dyalog APL for DOS/386	995	Second generation APL for DOS.Runs in 32-bit mode, supports very large workspaces. Unique "window-based" APL Development Environment and oSM Screen Manager. Requires 386/486 based PC or PS/2, at least 2Mb RAM, EGA or VGA, DOS 3.3 or later.
	Dyalog APL for Unix	995-12,000	Second generation APL for Unix systems. Available for Altos, Apollo, Bull, Dec, HP, IBM 6150, IBM RS/6000, Masscomp, Pyramid, NCR, Sun and Unisys machines, and for PCs and PC/2s running Xenix or AIX. Oracle interface available for IBM, Sun and Xenix versions.
I-APL Ltd	I-APL/PC or RML Nimbus	4.50	ISO conforming interpreter. Supplied only with manual. (see 'Other Products' for accompanying books)
	I-APL/BBC	4.50	As above
	I-APL/Archimedes	4.50	As above
IBM UK	IBM PC APL2	348	APL2 for the IBM PC. Program 5799-PGG/1, PRPQ number RJ-0411. From all IBM dealers.
MicroAPL	APL.68000 Level I	2000	First generation APL with numerous enhancements. Multi-user version (Unix, Mirage, MCS).
	APL.68000 Level II	2500	Second generation APL. Nested arrays, user defined operators, selective specification etc. Multi-user version (Unix, Mirage, MCS)
	APL.68000 Level I		
	Mac, ST, Amiga, QL	87	First generation APL. Single user, full windowing interface, software floating point support.
	Mac, Amiga	260	First generation APL. Single user, full windowing interface, hardware floating point.
	APL.68000 Level II		
	ST	170	Second generation APL. Full windowing interface, software floating point support.
	Amiga	260	Second generation APL. Full windowing interface.Hardware and software floating point support.
	Mac	520	Second generation APL. Full windowing interface.Hardware and software floating point support.

	APL*PLUS Rel 10	450	
	APL*PLUS II V 3.0	1395	
Reuters Ltd	SHARP APL	poa	for IBM mainframes
Univare	APL*PLUS/PC	495	STSC's full feature APL for IBM PC/XT/AT, Compaq, Olivetti.
	Run-Time	call	Closed version of APL*PLUS/PC which prevents user exposure to APL
	APL&PLUS/UNIX	call	STSC's full feature APL for UNIX based computers
	APL*PLUS II	call	STSC's full feature APL for 386 machines.

APL VISUAL DISPLAY UNITS

COMPANY	PRODUCT	PRICES(£)	DETAILS
APL People	IBM & compatibles	poa	IBM and 'budget' APL VDUs - monochrome/colour/graphics.
Dyadic	IBM 3151	599	Monochrome APL/ASCII vdu with APL keyboard. Supports downloaded Dyalog APL font.
	IBM 8154	1,228	Colour APL/ASCII vdu with APL keyboard. Supports downloaded Dyalog APL font.
General Software	Mellordata	poa	
Shandell	HDS3200/10 APL	965	15" screen, 8 page memory, windows, 80/132 columns, full overstrike, Multi-host multi-session support, ANSI X3.64, DEC VT100, VT220, Tektronix 4010/4014 1024 x 390 resolution.
	HDS3200/25 APL	1065	As above plus switchable 25 or 50 line screen. 75 Hz refresh. Resolution 1024 x 780 in graphics mode.
	HDS3200/35 APL	1265	As HDS3200/25 plus local pan & zoom.
	HDS3200/5C APL	1395	14" colour monitor. 75 Hz refresh, 8 or 18 colours from palette o 256. Display memory 96 lines of 80 or 132 columns. APL processing & keyboard with full overstrike. Windows, multi-host, multi-session support. ANSI X3.64, DEC VT220, VT100, VT52 emulation.

APL PRINTERS

COMPANY	PRODUCT	PRICES(£)	DETAILS
APL People	Epson series	200	Inexpensive dot-matrix and NLQ printers
	Quen-data & Qume etc	500	Daisy-wheel printers
Dyadic	Various	poa	Range of APL printers available.
MicroAPL	Datasouth DS180+	1,195	See Datatrade entry
	Phillips GP480	2,137	Matrix printer with letter & draft quality and APL (480 cps).
	Qume Letterpro20	549	APL/ASCII Daisy-wheel printer

APL PACKAGES

COMPANY	PRODUCT	PRICES(£)	DETAILS
Active Workspace Ltd	Syndicate Manager	poa	Lloyd's managing agent's syndicate / company accounting system. Stamp & Personal accounts (inc. Run offs)
APL-385	APL-385	50(PC),125(mf)	including ...
	FSM-385		Screen development
	DRAW-385		Screen design
	DB-385		Relational W.S.
	GEN-385		Miscellaneous Utilities

APL Software Ltd (mainframe)	RDS	poa	Relation Data Base System
	IPLS	poa	Project Management System
	REGGPAK	poa	Regression Analysis Package
(microcomputer)	POWERTOOLS	295	Assembler written replacement function for commonly used CPU-consuming APL functions, includes a Forms Processor.
	REGGPAK	poa	Regression Analysis Package
	RDS	990	Relational Database System
Cocking/Drury (for VSAPL)	E'MENTS & SHAREFILE	poa	Component files, quad- functions & nested arrays for VSAPL under VM/CMS & MVS/TSO
	COMPILER	poa	The First APL compiler
	FILEPRINT	poa	Print APL component files
	FILECONVERT	poa	Converts non-APL files to APL
	FILEMANAGER	poa	Extends APL primitives to database management
	TOOLS + UTILITIES	poa	APL Software development tools
(for APL2)	DATAPORT	poa	Information Centre spreadsheet incorporating data exchange between APL, FOCUS, IFPS, SAS, APL/DI, ADRSII, Lotus123, Visicalc, Multiplan & DIF
	SHAREFILE/AP	poa	STSC's shared access component file system for APL2. Comparable to all APL*PLUS file systems: multi-user storage of APL2 arrays with efficient disk usage
	FMT	poa	Full featured FMT for APL2
	WSDOC	poa	Workspace documentation utilities
	FILEMANAGER	poa	Extends APL primitives to database management
(for PC's)	APL*PLUS PC Tools	275	Utilities including: RAM disk, full screen data entry, menu input, report generation, exception handling and games.
	IRMA Module	90	327x IRMA support.
	FIN & STAT. LIBRARY	250	Financial & Statistical routines
	SPREADSHEET MGR	150	APL-based spreadsheet for APL*PLUS/PC. Cell arithmetic; transfers to ASCII & Lotus
H.M.W.	4XTRA	poa	Front-end Foreign Exchange dealing / pos keeping
	Arbitrage	poa	Arbitrage modelling
	Basket	poa	Basket currency modelling
	Menu-Bar	poa	pull-down menu for APL*PLUS/PC
HRH Systems	APL Utilities	poa	PC Utilities including: APLMAC (windows); Unlock (unlocks functions in .AWS); DTEX (text and spreadsheet imp/exp). Mostly available in English or French.
Impetus Ltd	<i>Impetus</i>	poa	Corporate Modelling and Reporting System.
INFOSTROY	Russificator	poa	Drivers and documentation for use with APL*PLUS/PC system and other STSC software with Cyrillic alphabet (PC).
Interprocess	IEDIT	1900-3200	Full screen APL2 editor with immediate APL execution, and a full-screen debugger
(mainframe)	AFM	8200-9800	High performance component and keyed file system (VS APL and APL2)
	Format	1650	A QuadFMT data formatter for VS APL and APL2
	FSM124	1650	AP124 programming for APL applications without GDDM (APL2)
	PowerCode	1300	External functions for APL2
	CALL/AP	3000	for calling non-APL programs (VS APL and APL2)
	UCF	1800	Inter-user data transfer for VM users via IUVC

(PC)	AFM	115	Single user component and keyed files for APL2/PC	
Mercla	STATGRAPHICS 4	684	Integrated statistics/graphics system for the PC. Now with macros. Bulk and educational discounts available.	
	Upgrade 3 to 4	215		
	Upgrade pre 3 to 4	375		
	MICROSPAN	250	Comprehensive APL tutor	
	LOGOL	poa	Logistics management system for PC and 386. Sales Forecasting, Inventory Control, Master Scheduling, Distribution Requirements, Planning etc	
	TWIGS		A modular library of tools to teach and explore state-of-the-art materials management concepts.	
	T	299	Time series forecasting	
	W	99	Warehouse replenishment	
	I	199	Inventory Management	
	G	99	Grouping requirements into ECOQ's	
	S	99	Scheduling production/purchasing	
			599	All 5 modules above
			4999	All 5 modules site licence
	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	
APL-CALC		400	APL spreadsheet system	
MicroPLOT/PC		250	For APL*PLUS/PC product	
MicroSPAN/PC		250	APL self instruction for APL*PLUS/PC	
	STATGRAPHICS Rel 4	590		
Reuters Ltd	GLOBAL LIMITS	poa	Exposure management for banks	
	IPSA/CONNECT	poa	Mainframe to micro link	
	MAILBOX	poa	Electronic Mail	
	MAILBOX/PC V.2	75	Full screen front end to IPSA mailbox	
	upgrade to V.2	15		
	NEWSFLASH	poa	Real time message exchange	
	VIEWPOINT	poa	4GL - info centre product	
Uniware (mainframe)	STSC's ENHANCEMENTS	poa	Quad-functions & nested arrays for IBM VSAPL under VM/CMS and MVS/TSO	
	STSC's SHAREFILE	poa	component files for IBM VSAPL under VM/CMS and MVS/TSO and for IBM APL2	
	PROGRAMMER TOOLS & UTILITIES	POA		
	FILEPRINT	poa		
	FILESORT	poa		

	FILECONVERT	poa	
	FILEMANAGER(EMMA)	poa	STSC's database package
	EXECUCALC	poa	Mainframe spreadsheet compatible with VISICALC and part of LOTUS 1-2-3 under VSAPL(VM or TSO)
(microcomputer)	STATGRAPHICS	poa	Statistics and graphics for PCs
	STATGRAPHICS UNISTAT	poa	An add-on module to STATGRAPHICS: Data analysis software.
	APL*PLUS TOOLS		
	-VOL1	poa	Incl. 327 AE IRMA support, RAM disk, full screen data entry, menu input, report generation, games
	-VOL2	poa	Incl. File documentor, screen editor, exception handling.
	SPREADSHEET MNGR	poa	poa APL spreadsheet with built-in ASCII, LOTUS and SYMPHONY interfaces.
	APL*PLUS/PC FIN & STAT.LIBRARY	poa	Collection of financial and statistical utilities.
	POCKET APL	poa	Smaller version of APL*PLUS/PC.
	UNIASM	poa	Collection of assembler routines for APL*PLUS/PC users.
	UNITAB	poa	APL*PLUS/PC spreadsheet-like data entry and validation system.
	The APL DEBUGGER	poa	First released APL*PLUS/PC debugger.
	APL2C	poa	Interface between APL*PLUS/PC and DATALIGHT C language
Warwick University	BATS	250	Menu driven system for time series analysis and forecasting using Bayesian Dynamic modelling. Price is reduced to £35 for academic institutions.
	FAB	free	Training program for the above.

APL CONSULTANCY

(prices quoted are per day unless otherwise marked)

COMPANY	PRODUCT	PRICES(£)	DETAILS
Active Workspace Ltd	Consult'	poa	PC Based APL system design, programming and implementation.
Adfee	Consultancy	poa	Development, maintenance, conversion, migration, documentation, of APL products in all APL environments
APL People	Consultancy	poa	Consultants available at all levels, with experience in: VS APL, APL*PLUS, APL2, Sharp APL, Dyalog APL, APL6800, C/Unix, TSO/MVS, VM/CMS, graphics, Operational Research etc. Expertise in APL system design, project management, prototyping, financial applications, decision support systems, MIS, links to non-APL systems, documentation, etc.
Buckland Management Systems	Consultancy	poa	Business and Technical systems in commerce and industry - designing, programming and implementing applications.
Carnacho	Consultancy	poa	Specialising in programming & manual writing.
Chapman	Consultancy	150-300	24-hour programmer: APL, C, assembler, graphics; PC, mini, mainframe, network.
Cocking/Drury	Consultancy	175-275	Junior consultant
		275-350	Consultant
		300-450	Senior consultant
		400-600	Principal Consultant

		450-750	Managing consultant
Peter Cyriax	Consultancy	100-150	Junior Consultant
		120-200	Consultant
		160-300	Senior Consultant
Delphi	Consultancy	poa	APL system development on mainframes and micros.
Dyadic	Consultancy	poa	APL and Unix system design, consultancy, programming and training.
E & S	Consultancy	poa	System prototyping; all types of information system, engineering software, graphics and decision support systems APL*PLUS/PC, APL2, Dyalog APL
General Software	Consultancy	from 120	
Michael Hughes	Consultancy	poa	Consultant with 10+ years experience with various APL interpreters and C.
H.M.W.	Consultancy	poa	System design consultancy, programming. HMW specialize in banking and prototyping work.
Ian A. Clark	Consultancy	poa	Computer-based Information Systems Implementation where acceptance is critical. APL on PC and Macintosh. Human Factors of HCI; novice ease of use; online assistance; training courses; distance-learning materials.
INFOSTROY	Consultancy	poa	Localization of APL software for the Soviet Union software market
Intelligent Programs Ltd	Consultancy	175-350	Systems development, enhancements, support.
	Documentation	150-250	Preparation of new manuals, rewriting of existing materials.
	Training	150-250	Training for APL experts through to non-technical system users.
Mercia	Consultancy	poa	APL*PLUS & VSAPL consultancy.
MicroAPL	Consultancy	poa	Technical & applications consultancy.
M.T.I.C.	Consultancy	240-500	Business analysis and APL consultancy
Parallax Systems Inc	Consultancy	\$750	Introductory APL, APL for End-user & Advanced Topics in APL
QB On-Line	Consultancy	250	Specialising in Banking, Financial & Planning Systems.
Reuters Ltd			
	Consultancy	poa	Consultancy & support service world-wide.
Rochester Group	Consultancy	poa	Specialise in MIS using Sharp APL
Rex Swain	Consultancy	poa	Independent consultant, 15 years experience. Custom software development & training, PC and/or mainframe.
Wickliffe Computer Ltd	Consultancy	poa	System design, consultancy, programming and documentation. Especially project management and decision support systems

OTHER PRODUCTS

COMPANY	PRODUCT	PRICES(£)	DETAILS
Adfee	Employment	poa	Contractors and permanent employees
APL People	Employment Agency	poa	Permanent employees placed at all levels. Contractors supplied for short/long-term contracts, supervised or unsupervised. Executive Search service available.
HMW	Employment	poa	Contractors and permanent employees placed.
I-APL Ltd	An APL Tutorial	2.50	45pp by Alvord & Thomson
	An Encyclopaedia of APL (2nd Edn)	5.50	228pp by Helzer
	APL in Social Studies	2.50	36pp by Traberman
	I-APL Instruction Manual (2nd Edn)	2.50	55pp by Camacho & Ziemann

	APL Programs for the Mathematics Classroom (Springer-Verlag)		
		14.50	185pp by Thomson
	** Please add one pound packing charge per order **		
Reuters Ltd	Productivity Tools	poa	Utilities for systems, operations, administration & analysts; auxiliary processors, comms software, International network.
	Databases	poa	Financial, aviation, energy and socioeconomic.
ISI	Tangible Math	\$19	An APL Approach to Math - Shareware, includes base Sharp APL
	J	\$24	Dictionary APL simplified and enhanced - Shareware (Mac,PC)
	SharpAPL/PC	\$74	Registered Shareware and Reference Manual
	ISI APL	\$99	Improved APL/PC - enhancements, performance, large workspaces
Renaissance Data Systems	Booksellers		The widest range of APL books available anywhere. See Vector advertisements.

OVERSEAS ASSOCIATIONS (Sample entries)

GROUP	LOCATION	JOURNAL	OTHER SERVICES	Ann.Sub.	Visa/Mcd
APL Bay Area	USA N. California	APLBUG	Monthly Meetings (2nd Monday)	\$15	N/N
Dutch APL U.G.	Holland	-	Mini-congress, APL ShareWare Initiative		
APL Club Austria	Austria	-	Quarterly Meetings	200AS per person, 1000AS per company	N/N
Sydney APLUG	Sydney, Australia	Epsilon	Monthly Meetings		

VENDOR ADDRESSES

COMPANY	CONTACT	ADDRESS & TELEPHONE No.
Active Workspace Ltd	Ross D Ranson	Moulsham Mill Centre, Parkway, Chelmsford, Essex, CM2 7PX. Tel: (0245) 252414 ext.240
Adfee	Bernard Smoor	Dorpsstraat 50, 4128 BZ LEXMOND, Netherlands. Tel: 31.3474.2337, fax: 31.3474.2342
APL 385	Adrian Smith	Brook House, Gilling East, York. Tel: 04393-385
APLBUG	Jorge Mezel	117 East Creek Dr., Menlo Park, CA 94025, USA
APL Club Austria	Erich Gail	IBM Osterreich, Obere Donaustrasse 95, A-1020 Wien, Austria
APL People	Jill Moss	The Old Malthouse, Clarence St, BATH, BA1 5NS. Tel: 0225-462602
APL Software	David Ails Jill Moss	The Old Malthouse, Clarence ST, BATH, BA1 5NS. Tel: 0225-462602
Buckland Management Systems	John Buckland	Westwood, 19 Grange Road, Camberley, Surrey, GU15 2DH Tel: 0276 684327
Anthony Camacho		2 Blenheim Road, St. Albans, Herts AL1 4NR. Tel: St. Albans (0727) 880130
Paul Chapman		18, Trevelyan Road, London, SW17 9LN Tel: 081-767 4254
Cocking & Drury Ltd.	Romilly Cocking	180 Tottenham Court Road, LONDON, W1P 9LE Tel: 071436 9481 Fax: 071-436 0524
Datatrade Ltd.	Tony Checksfeld	38 Billing Road, Northampton, NN1 5DQ. Tel: 0604-22289
Delphi Consultation	David Crossley	Church Green House, Stanford-in-the-Vale, Oxon SN7 8LQ. Tel: 0367 710384
Dutch APL U.G.	Bernard Smoor (Sec)	Postbus 1341, 3430BH Nieuwegein. Tel: 03474-2337

Dyadic Systems Ltd.	Peter Donnelly	Riverside View, Basing Road, Old Basing, Basingstoke, Hants RG24 0AL. Tel: 0256 811125 Fax: 0256 811130
E & S Associates	Frank Evans	19 Homesdale Road, Orlington, Kent BR5 1JS. Tel: 0689-24741
General Software Ltd	M.E.Martin	22 Russell Road, Northholt, Middx, UB5 4QS. Tel:081-864-9537
H.M.W. Trading Systems	Stan Wilkinson	Hamilton House, 1 Temple Avenue, Victoria Embankment, LONDON EC4Y 0HA Tel: 071-353 4212 Telex: 926604 HAMHSEG Fax: 071-353 3325
HRH Systems	Dick Holt	Box 4496, Silver Spring, Maryland 20904
Michael Hughes		28 Rushton Road, Wilbarston, Market Harborough, Leics., LE16 8QL. Tel:0536 770998
Ian A. Clark		9 Hill End, Frosterley, Bp. Auckland, Co. Durham DL13 2SX Tel: 038852-7190
I-APL Ltd	Anthony Camacho	2 Blenheim Road, St. Albans, AL1 4NR. Phone 0727-860130 for queries, order forms, bulk orders
	J C Business Services	56 The Crescent, Milton, Weston-super-Mare, Avon, BS22 8DU
IBM UK Ltd	Nat.Enq. Centre	414 Chiswick High Rd, London W4 5TF Tel: 081-747 0747
Impetus Ltd	Cedric Heddle	Rusper, Sandy Lane, Ivy Hatch, SEVENOAKS, Kent TN15 0PD Tel: 0732-885126
INFOSTROY	Alexei Miroshnikov	3 S. Tulenin Lane, Leningrad 191186 USSR. Tel:812-238-6392 Fax:812-319-9709
Interprocess Systems	Stella Chamberlain	9040 Roswell Road, Suite 690, Atlanta, Georgia 30350-1131 Tel: (404) 992-8400
Intelligent Programs Ltd	Mike Bucknall	Unit 7, Hermitage Court, 6-10 Sampson Street, London E1 9NA Tel:071-481-4813
ISI	Eric Iverson	33 Major Street, Toronto, Ontario, Canada M5S 2K9 Tel:(416) 925-6096
	Orders	3512 Cameron Mills Road, Alexandria, Virginia, USA 22305-1103 Tel:(703) 548-1799
Mercia Software Ltd.	Gareth Brentnall	Aston Science Park, Love Lane, Birmingham B7 4BJ. Tel: 021-359 5096
MicroAPL Ltd.	David Eastwood	South Bank Technopark, 90 London Road, LONDON SE1 6LN Tel: 071-922 8866
M.T.I.C.	Ray Cannon	21 Woodbridge Rd, Blackwater, Camberley, Surrey GU17 0BS Tel: 0252 874697
Peter Cyriax Systems	Peter Cyriax	12 Gloucester Place, London W1H 3AW Tel: 071-935 2983
Parallax Systems Inc.	Kevin Weaver	Avery Road, Box 319, Garrison, NY 10524, U.S.A. Tel: 914-424-4265
QB On-Line Systems	Phillip Bulmer	5 Surrey House, Portsmouth Rd Camberley, Surrey, GU15 1LB. Tel: 0278-20789
Renaissance Data Systems	Ed Shaw	P.O. Box 20023, Park West Finance Station, New York, NY 10025-1510, U.S.A.. Tel: (212)864-3078
Reuters Ltd	Paul Jackson	7th Floor B Block, Coventry Point, Market Way, Coventry CV1 1EA Tel: 0203 258562
The Rochester Group	Robert Pullman	50 S. Union St., Rochester NY 14607, U.S.A. Tel:716-454-4360 or 716- 454-4641
Shandell Systems Ltd.	Maurice Shanahan	Chiltern House, High Street, Chalfont St. Giles, Bucks., HP8 4QH. Tel: 02407-2027. Fax:02407-3118
Sugar Mill Software Corp.	Lawrence H. Nitz	1180 Kika Place, Kailua, Hawaii 96734 Tel: (808) 261-7536
Rex Swain		8 South Street, Washington, CT 06793, U.S.A. Tel:203-868-0131
Sydney APLUG	Rob Hodgkinson	GPO Box 1425, Sydney, NSW 2001, Australia
Uniware	Eric Lescasse	15 Rue Erlanger, 75016 Paris, France. Tel:(1) 45-27-20-61. Fax:(1)45-27- 20.61. Telex: 648348F UNIWARE
Wickliffe Computer Ltd	Nick Telfer	78 Victoria Rd., Whitehaven, Cumbria, CA28 6JD. Tel:0946-692588
Warwick Univ.	Prof. Jeff Harrison	Dept of Statistics, University of Warwick, Coventry, CV4 7AL Tel:0203-523369
Zark Incorporated	Gary A. Bergquist	53 Shenpsit St, Vernon CT 06066

Book Review: Programming in J

reviewed by Keith Smillie

Iverson, Kenneth E., 1991. *Programming in J*.
Iverson Software Inc., 33 Major Street, Toronto M5S 2K9, Canada.
69 pp., \$40.00 + 7% GST.

Programming in J begins as follows:

"This book is an introduction to programming in J, a language available as shareware on a wide variety of computers. Although the primary purpose of study may be to learn to *write* programs and systems for use on a computer, *reading* is an interesting and efficient aid to gaining fluency in writing. Much emphasis is therefore given to exercises in reading."

Further on the first page we read that:

"A prospective programmer may feel that learning to read a language is a costly detour on the way to writing, but imagine the analogous case of a person unwilling to learn to read English because he is impatient to learn to write. Moreover, reading alone is a very useful skill; it allows one to read what others have written, in order to quickly grasp otherwise complex procedures."

This monograph, then, is intended to be used primarily as a reader to introduce persons to J. It supplements the *ISI Dictionary of J*, and the first chapter has been designed "to encourage and reinforce reference" to the dictionary.

The first five of the six chapters are mainly about reading J. Chapter 1, "Introduction", gives an overview of the language. Each of the fifteen sections introduces a topic in J, e.g., nouns and pronouns, verbs and proverbs, conjunctions, and most of the sections contain exercises to read and execute on the computer. Chapter 5, "Experiments", contains further examples for each of the sections of Chapter 1. Chapters 2, 3 and 4, entitled "Adverbs", "Conjugations" and "Verbs", respectively, continue the discussions of these topics begun in Chapter 1. Only in Chapter 6, "Writing Programs", does writing take precedence over reading. This last chapter discusses the use of J in a variety of topics including the preparation and presentation of tables of numerical data, construction of graphs and barcharts, and polynomial manipulation.

Programming in J is attractively printed in a 5½" by 8½" format. The inside back cover gives as Appendix A a table of the fifty-odd foreign conjunctions used for communicating with the host facility, and the outside back cover gives the

language summary as is apparently customary in all of the publications from ISI Software.

Persons familiar with APL will recognize Ken Iverson's terse, even spartan, style. The following are a few short quotes to illustrate his approach to learning J: "Do not be stopped by the discussion of *rank*; it might be better to defer the matter until rank is used in Section I of this chapter"; "Read Part II.B (Verbs) and enter all examples, but do not become paralyzed by difficulties; it may be best to review them again later"; "Read all of Appendix A lightly and continue with the following experiments"; and "Review Appendix A and experiment with any facilities that look interesting." We should not overlook his one-sentence summary of structured programming given in the last chapter: "Use structured design, constructing each program from a few sentences that themselves employ component functions, which may in turn be constructed of generally-useful components."

Rather than attempt to review the contents of Programming in J let us consider a paragraph or two which might be considered typical of the style and the demands placed on the reader. The following is the discussion of the verb permute given in Section G of Chapter 4:

If $p = .\ 0\ 2\ 4\ 3\ 1$, then $p\{b\ 'abcde'$ provides a permutation of the items of b , and p is said to be a *permutation vector*. Read the first paragraph of the definition [in the Dictionary] of the dyadic case of permute ($C.$), and enter $p\ C.\ b$ to verify that the result is the same as that of $p\{b$. Also verify that $3\ 1\ C.\ 1.\ #b$ yields p .

Read the next paragraph of the definition and reconcile it with the results obtained above. Then examine the following dialogue and read the entire definition of $C.$ (including the monadic case):

```

]c=.C.p
+++++-----+
|0|3|4 1 2|
+++++-----+
  c C. b
acedb
  c C.i.#b
0 2 4 3 1

```

EXERCISES

5. Define $f = .\ 1.\ \&!A.\ 1.$ and experiment with $f\ n$ for various values of n .
6. Do the exercises on permutation from Chapter 6.

Programming in J must be read slowly and thoughtfully. Most sections need to be read repeatedly and the examples and exercises experimented with on a computer. The reader who is encountering J for the first time could easily become discouraged. The best advice in reading this monograph can probably be stated in the author's own words quoted above: *Do not become paralyzed by difficulties*. Progress will almost certainly be slow initially, but perseverance in reading together with working on simple problems of one's own with the examples given in the text as models will eventually give a fluency in using J.

Today many conventional programming languages together with spreadsheets, data bases and statistical packages in implementations which are increasingly easier to use compete for our attention. However, this reviewer experienced the same excitement on being introduced to J as he did when he began working with APL twenty-five years ago, and is obtaining the same satisfaction in "mastering J", to use a colleague's phrase, as he did when he was working with APL. He considers J to be an important step in the development of programming languages.

In his opinion *Programming in J*, together with the *ISI Dictionary of J*, are indispensable in understanding J and investigating its place in the computing world of the late twentieth century.

Keith Smillie
Department of Computing Science
University of Alberta
Edmonton, Alberta T6G 2H1

APL and J: Some Benchmarks

by Malcolm Rigg

I first learned APL via a self-teach set of lessons on APL/360 in the early 70s and still have some faded notes from 1974 sellotaped into two A4 binders. I was at the time fascinated by the notation and determined to master it. What started as an experiment in my IBM days at Hursley Labs continued when I joined ICL Dataskil at Reading in 1977 and shortly after went out to join the ICL EC project in Luxembourg. There an ICL Mainframe was being introduced and work transferred to it. As this included APL I was able, as part of my job, to use APL (some people will remember the 2900 APL I'm sure), write code on a variety of machines, teach APL courses (in both English and French), and of course enjoy being a good European in the heart of the EEC. Teaching APL introduced me to a large number of fonctionnaires in both Luxembourg and Brussels, some of whom have remained contacts and friends over many years. Regrettably after nearly six years I and the family returned to the UK.

Since then APL has definitely been on the back burner (there being no easily accessible APL at work) until I came across first I-APL and then J. I have been able to experiment a little with both of these languages and offer my experiences in performance comparisons using the Adrian Smith "benchmarks". At home I have an XT which has been extended with an 8087, V30 replacing 8086, 2Mb expanded memory and two SCSI drives giving me a total of 290Mb filestore. Although I have repeated some of the performance tests at work where I have a 386 and 486 machine, I have not included them all. My son's Archimedes 410 has 4Mb mainstore, 100Mb HD and an ARM3 running at 30MHz and has been used to test the 16 and 32 bit versions of I-APL. I wish I had an APL able to exploit the power of this machine.

Having coded the Adrian Smith "benchmarks" in APL*plus, Sharp APL, I-APL, and J, I looked around for a more meaty test and settled on an APL implementation of the N Queens problem. Starting from a classical APL approach I have been attempting to rewrite my first non-optimised APL solution in J - quite an interesting task and a reasonable test of J (and me) as my APL original is heavily recursive. Since starting I have received a copy of the APL algorithm written by Roger Hui; its so much faster than mine that I shall think hard about a version in J when I have fully understood his APL code. However my version does give a solution for 10 queens where his gives me wsfull!

I am looking forward to the day when J settles down and it becomes possible to write code in workspaces of a reasonably large size. What would also be useful would be an easier way to develop functions; I still find that use of \$. is difficult to grasp. Perhaps a summary example explaining how to extend a multi-line function into a conditional loop structure would help. For the moment I find that to write J takes much more intellectual effort and there is definitely a higher hurdle to jump before being able to use J than APL.

I received from Anthony Camacho the J 3.4 for the Archimedes with its rather nice integration into the desktop environment. The performance figures using the same benchmark are given below. I must also admit that the section Tutorials in the new version of the "ISI Dictionary of J" goes a long way towards explaining by example.

On balance I like using J, have a few dislikes like syntax with mandatory spaces and workspace restrictions, and hope that my wishlist for improvements will come true in 1992.

The Benchmarks

Here is a summary of the Adrian Smith 'benchmark' run on several APLs. The PC-based results are for an XT at 9.6MHz with 8087 maths co-processor (all times in ms). For comparison the I-APL results on both the PC/XT and an Archimedes are included as the last column. In some cases several runs have been averaged as the clock resolution is not that good. A few tests failed for reasons indicated.

Test	APLplus/PCSharp (6.1)	APL v17	J v20	J 2.8	J 3.3	I-APL PC/XT	J-Arc ARM2	J-Arc ARM3	I-APL(32) ARM3		
1	165	493	494	675	675	5930	1180	360	460		
2	30	110	110	55	55	800	30	30	70		
3	140	1154	1153	10000	7600	6050	3160	2080	500		
4	700	1400	1318	824	1900	6200	2750	880	520		
5	165	355	330	110	110	5000	120	40	630		
6	385	2087	2087	9500	7050	14950	2120	1370	2050		
7	1300	4557	4559	limit	err	limit	err	59430	2070	580	4580
8	30	2140	2142	360	385	990	30	10	100		
9	1620	2527	2526	1483	1480	35150	270	60	2560		
10	1810	355	329	2340	2400	111270	650	110	8430		
11	60	163	140	21200	21620	19400	5190	2950	1240		
12	100	110	55	wsfull	wsfull	27400	fail	fail	2380		
13	150	2746	2417	2280	2200	2860	690	410	430		
14	1370	25376	21751	21350	20300	25380	5240	3230	4010		
15	14170	254207	217500	212100	201000	250470	51050	31320	39700		
16	30	660	659	2270	2308	24320	650	320	2030		
17	12750	12193	12160	58500	51200	limit	err	11840	6080	30580	

The essential difference between the J2.8 and J3.3 script is the use of "." instead of ".." in an inner product. A space must occur before this delimiter; an unfortunate fact for a specification which is hopefully otherwise whitespace independent. The function fn is defined with "." or "·" in J3.3, J2.8 respectively. The Archimedes ran in mode 12 with no other multi-tasking applications.

The Original 'Smith' Benchmarks (in APL and in J)

```
V←1000ρ 4 5 6 0
M←100 10 ρ 2.5 6.7 0 0 8 -5.7
S←10000ρ 'THE CAT SAT ... MAT'
S[999+16]←'ADRIAN'
I←100?1000
B←10000ρV=6
FF←800 5 ρ'TR HG HJ ...'
FF[601;]←'EJD..'
```

```
v=.1000$4 5 6 0
m=.100 10$2.5 6.7 0 0 _5.7
s=.10000$'the ... '
s=. 'adrian' (999+1.6) }s
i=.100?1000
b=.10000$v=6
ff=.4000$'tr ... '
ff=.800 5$'ejd..' (3000+1.5)}ff
```

```
∇ FN N;CT;X
[1] CT←0
[2] LP:→(N<CT←CT+1)↑EXIT
[3] X←+/ 2 3 ρ16
[4] →LP
[5] EXIT:'DONE'
∇
```

APL Expression	Type	J Expression
1. +/V+V	⌈ INT	+/v+v
2. +/V[I]	⌈ INT	+/i{v
3. V+.×V	⌈ REAL	v+/ .*v
4. +/V*2	⌈ REAL	+/v^2
5. X←+/M	⌈ REAL	x=.*+m
6. V+.×,M	⌈ REAL	v+/ .*,m
7. +/S='T'	⌈ CHAR	+/s='t'
8. 'FAT'εS	⌈ CHAR	'fat'ε.s
9. X←Sε'FAT'	⌈ CHAR	x=.s ε. 'fat'
10. X←B∨B∧~B	⌈ BOOL	x=.b +. b *. -. b
11. ∧/B	⌈ BOOL	*./b
12. X←∨\B	⌈ BOOL	x=.*./\b
13. FN 10	⌈ FN	fn 10
14. FN 100	⌈ FN	fn 100
15. FN 1000	⌈ FN	fn 1000
16. X←FF∧.= 'EJD..'	⌈ IP	x=.*./ff='ejd..'
17. X←FF∧.=⊆20 5ρ'EJD..'	⌈ IP	x=.*./ff.= :20 5\$'ejd..'

RECENT MEETINGS

This section of VECTOR is intended to document the seminars given at recent meetings of the association; it is of particular value to members who live away from London. It also covers other selected events which may be of general interest to the APL community.

If you would like to speak at one of the regular British APL Association seminars, please ring the Activities Officer (address on inside back cover) who will respond enthusiastically to your offer.

Panel Design: An APL Programmer's Toolkit

by David Crossley

The value of any aid in designing and generating panels for screen display is readily appreciated by those involved in serious applications development. Using the basic screen facilities of your version of APL may be fun at first, but soon the sheer repetitive nature of screen interface handling becomes tiresome. Most programmers will develop a few macros and this helps. Most suppliers also provide utility workspaces to design panels, or forms as they are often called. However, in my view a fully integrated design toolkit is the only real answer.

The subject of this article is such a package. It incorporates well tried and tested ideas gleaned from many years working with a very good panel design facility in a mainframe environment. The contribution that a good facility makes towards programming productivity should not be underestimated. My experience suggests 40-50% savings in screen-intensive applications, and most of them are. There are also excellent spin-offs. It is easier to provide a consistent, corporate look and feel across (and within) applications. It is more likely that proper help facilities will be incorporated. Support for special features, such as sliders, pop-ups and pull-downs can be included. Panel design packages provide immediate potential for automatic documentation - never a strong point for any programmer I have worked with. But importantly from the systems design viewpoint, the supporting facilities can provide the complete basis for full interface handling - the screen display, keyboard and mouse control.

The origins of this panel design package go back about two years, though the concepts considerably pre-date that time. After many years using mainframe APL, I started work in a relatively greenfield environment (for the company) with PCs linked to a workstation under Unix. There were few support facilities. Though some had been migrated from the mainframe, the important area of panel design was unsupported, partly because the APL in use was Dyalog with the recently introduced $\square SM$, an advanced screen management facility. My first task was to write a panel design package along with support functions, with Dyadic's useful though limited $\square SM$ design workspace as a starting point. This effort repaid itself many times over and soon migrated to other development areas within the company.

I have since developed various applications for 80386/80486-based personal computers running under DOS. The lack of proper mouse and full colour support common to most APLs within this environment has increasingly irked me, especially with MS Windows on the same machine, unhappily incompatible with APL. Inspired by recent Vector articles [Cannon 91;Smith 90], I researched the use of DOS interrupts directly from APL. The results were sufficiently impressive to warrant a complete re-write of the panel design package to incorporate full support, and many additional features, within the software. I call this package the Delphi Panel Design Toolkit after my trading name.

What is a Panel Design Toolkit?

The first component of a toolkit is a mini-application to describe a panel, or form, and store it away somewhere for later use. This design tool allows different areas of the panel to be identified as "fields" and ascribes various attributes to each field, such as whether or not input is permitted, what sort of data it will contain, colours and so forth. There is a great deal of scope here for associating properties with fields that will simplify the programming task later; a function to validate user input is one such idea. Special fields might also be recognised, such as a menu bar now made so familiar by its acceptance in graphical user interfaces.

Many panel design packages effectively stop at this point without realising the great potential in terms of application interface control. This is a mistake. The second component should be the provision of a host of programming support utilities. Firstly, a "setup" utility to retrieve a saved panel, usually from a file, and display it with current application data. Secondly, a "read" utility to handle all user communications, whether by means of keyboard or mouse. And finally, a supporting cast of functions to allow the programmer to make changes on the fly in a controlled manner. For example, items on a menu bar might be temporarily disabled but remain visible; better done in cooperation with panel software.

The "read" utility is perhaps the single most powerful tool. It can handle not only pedestrian traffic, such as accepting input, refreshing the screen, scrolling fields and so on, but also more sophisticated tasks such as validating input, insisting on corrections for inappropriate entries, intercepting specific events such as help requests - the scope is tremendous. Each activity handled by the tool is a task less for the programmer to worry about. If an event can be handled in a standard way, let the tool do it. But allow other events to return control to the program for suitable action.

These are the philosophies embodied in the *Delphi Panel Design Toolkit*.

The APL Framework

Panel definitions are stored on an APL component file, normally associated with a single application. The *Panel Definition Tool* (see below) has its own panel file, and indeed the application code utilises its own services correctly. A file may be created using the *P_CREATE* utility, or else the tool can do the work when the first panel is saved. There is no naming convention imposed, but I find it useful to have a name of the form 'panxxxxx' for ease of identification.

The file structure is very simple. It has a directory which records panel names, short descriptions, creation timestamps and pointers to other components where the full definition for a panel resides. A component is reserved optionally to store red-green-blue colour scales that allows individual colours for an application to be customised - more on this later.

A panel definition variable is a nested vector with 4 items:

- the associated filename;
- the panel identification name;
- a nested structure that describes function (or other) key actions;
- a matrix with one row per field, describing attributes for that field.

This panel variable is defined in the workspace when required and is an argument to and returned in the result from almost all service functions. Its contents may be altered in the process. Programmers who wish to meddle directly with this variable, as they surely will, are provided with a set of subscript variables for items and columns of the main matrix, such as *FΔVIDEO* for the column containing the video (colour) attribute. This is a good general practice since it soothes some of the pains of future upgrades.

There are a few globals but not many. One holds the last-used panel file, allowing the filename to be elided in various function calls, i.e. omitted or represented as ''. Other globals relate to the mouse, keyboard and output translation, the latter being implementation specific.

Mouse Technology

For readers who are not familiar with the use of a mouse, and perhaps also for some who are, here is a short digression on the subject. Physically, a mouse has two or three buttons and plugs into a serial (or parallel) port to the computer. It translates roller ball movements into relative screen movements of a small pointer on the screen. To use a mouse, a driver program must be loaded into DOS. This can be done at any time but is usually included in the boot procedures. The load program specifies which protocol is to be used; in effect,

how many buttons are active. The norm is two active buttons (left and right), but three can be made active.

All sorts of mouse attributes are configurable, such as the pointer shape and colours, and the dynamic properties of its movement. Attributes, mouse button presses and movements are accessed through specific APL facilities, such as `MOUSE`, or through a set of DOS interrupts (the method used in my software). For full details refer to [Duncan 88].

Essentially, the programmer is interested in three parameters for controlled use of the mouse:

- the screen coordinates of the mouse;
- the combination of buttons that are currently pressed (including none);
- the perceived state of the mouse.

With a two-button mouse, there are 4 button combinations, *viz* left, right, left and right, and none. It is nice for left-handed people to have the option to swap left and right buttons, since the index finger is generally preferred for the main clicking task. Unfortunately, this must be supported in software.

The useful mouse states are single click, double click, and drag (button held down). Their detection requires constant monitoring using a short time interval to distinguish between states. This interval should be user-configurable (as should the mouse's dynamic properties) since some people react faster than others.

Thus a two-button mouse offers 10 different combinations of buttons and states (including no buttons pressed) and also the positional information on which to make program control decisions. In practice, only a small subset of events is acceptable. Conventionally, just the left button is live and click is the main activator. Double click is normally used to complete some activity.

The great virtue of the mouse is its enormous facility for rapid movement around the screen, making keystroke navigation even more ponderous than it seemed. From being a mouse-sceptic (before I used a mouse), I now applaud the "point-and-click" approach as effective and time-saving; it has opened up new horizons in the design of screen-based applications.

It's a Colourful World

Another digression. Most 80386/80486 machines these days have a Video Graphics Adaptor (VGA), and most text-based APLs grossly under-utilise its potential. Many of you will think that you are limited to the 8 or 16 rather lurid standard colours that are initially presented when you switch on. This is not the case.

The VGA is set up to mimic earlier graphics cards by default. Text modes still impose a limit of 16 simultaneous colours (8 standard plus 8 enhanced colours using the bright and blink bits for foreground and background respectively). But each of those 16 colours can be mixed individually to your own taste.

In fact, the VGA is sophisticated circuitry, differing by manufacturer, with very rapid switching facilities, especially if controlled directly. For text-based screens (as opposed to graphics), control at that level is unnecessary. The use of a few DOS interrupts accessible from most APLs is all that is required.

The 256 colour registers are split into 4 "pages" of 64 registers for text modes, only one page of which is used, and only a "palette" of 16 of those registers is used. Each register (or DAC register for digital-to-analogue-convector, to give it its full name) points to a colour mix. A mix is a combination of red, blue and green values on a scale from 0-63 that gives a specific colour. All 0's give black. All 63's give white. There are over 260,000 possibilities in between.

It is not really necessary to have a deeper understanding of the VGA beyond that described here to use the DOS interrupts. For the latter, and for a varied and readable book on the PC in general, I recommend [Norton 88].

The panel design tool incorporates a facility to customise colour palettes and the use of colours. This information is stored on the panel file for use over the whole application. Programmer tools are provided to switch palettes and to restore standard (or prior) palettes.

What's in a Panel?

Before reviewing the mechanics of the panel design tool, it may be helpful to look at an example of what it can achieve as the end-user would see it.

Figure 1 illustrates a very simple application which nevertheless incorporates several built-in features.

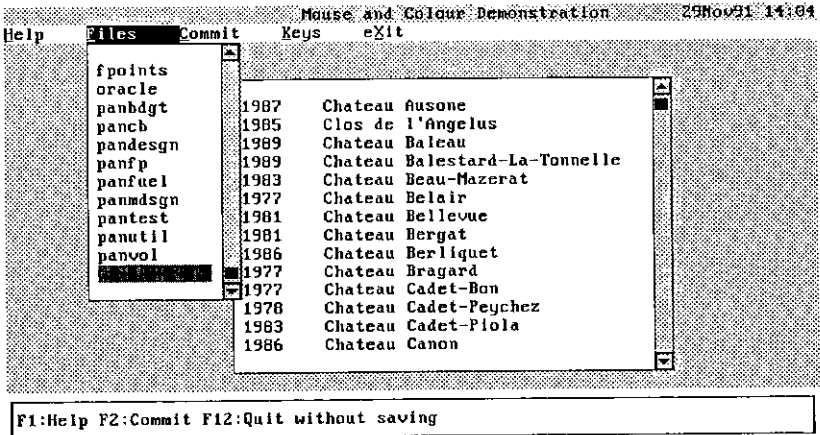


Figure 1: A panel with Files pull-down

On the top line there is a title and a date/time field. Then there is a menu bar. The uses for function keys are shown near the base of the screen. And two blocked areas have slider scroll bars to their right. These are all special field types supported without any further coding requirements from the programmer.

The title field takes its text from a specified variable and automatically centres it within the field width.

The date/time field is continuously updated at minute intervals.

A menu bar item may be activated by moving to the item and pressing *Enter*, pressing *Alt* with the underbarred character in the item word, or clicking on the item with the mouse. Each case is functionally synonymous. The item changes its video, normally white-on-black from black-on-white but the choice is left to the designer. Some standard actions are handled directly, such as Help (to display help for the current field or the entire panel) and Keys (to toggle the display of

function key box at the base of the screen on and off). Otherwise the event is returned to the program for appropriate action. In this example, the Files option has been requested and the action has been the display of a selection list of files. This pull-down is of course simply another panel overlaid on top of the main panel.

The function key box near the base of the screen is available as a prompt to the user. An action is invoked by pressing the function key (or indeed any key on the keyboard may be specified), or by clicking on the item with the mouse. Standard actions may also be associated with keys to be handled directly as for the menu bar. I always assign F1 to Help but keys can be arbitrarily assigned to suit house conventions. But normally, the event is reported back to the program for action. The display of the keys box can be toggled on and off. I generally assign F11 for this purpose, with the Keys option on the menu bar as an alternative.

Slider bars are associated during field definition and are then handled entirely by the toolkit software. Both vertical and horizontal sliders are supported. Several related scroll fields may be controlled by one slider, illustrated by the year and wine description fields in the example. These are strictly mouse operated though the data fields controlled by the slider may also be scrolled using movement keys.

Usage of slider bars is conventional. The ratio of the slider marker position to the slider length is kept in step with the ratio of the top-left visible element of the scroll data to its total length within rounding accuracy. Clicking within the slider moves the scroll data accordingly. Clicking on the top or bottom (left or right) box marked with a chevron moves the data by almost a screen window in that direction. By placing the mouse on the marker with the button held down, the marker may then be dragged with almost smooth scrolling of the data. A small box also appears giving the data row (or column) visible in the top-left corner. The slider marker is also updated when scrolled areas are moved by key actions.

The bottom line of the screen is used for error or informative messages if required. Messages are not generated by the panel utilities, but by the user program. If either of two variables (*ERROR* or *MESSAGE*) contains text, the message (on a striking red or a less attention-grabbing white background) is displayed on the bottom line by the "read" utility and cleared the next time round.

The panel tools do not impose any particular convention or layout on the designer. The entire screen may be used including the bottom lines. The special fields are not restricted to a specific place on the screen, with the exception of the keys box and message line. Both facilities can be switched off and remain unused

though the latter is in any case transient. Thus the designer is free to develop a personal style naturally taking account of house standards.

Let us now see how the main panel was defined.

The Panel Definition Tool

...Getting Started

This is a mini-application in its own right, utilising many of the facilities provided with the toolkit. It is initiated by monadic function *P_DESIGN*. Its argument can be:

- (a) a properly-formed panel variable,
- (b) a text file name and panel name,
- (c) an empty vector.

The result is a panel variable but it is "shy". This means that a result is returned if it is used, e.g. assigned to a variable name, but otherwise it is suppressed. Form (b) is usual but even here the file name may be elided to the last-used panel file. Normally panel definitions will be read from a panel file (for editing) and saved on completion, but the flexibility is there to edit and return a panel variable from the workspace, saving it en route if desired.

...Defining Fields

For a new panel, a blank screen is presented except for a list of function key uses near the base of the screen. The keys can be toggled off and on using F11. This is a completely empty panel with no fields.

A field is a rectangular box marked using the cursor movement keys after selecting the Create option, or more swiftly by creating a spot and then dragging a box anchored at one corner with the mouse. The field acquires default attributes of which more later.

Fields can overlay other fields, obscuring what lies beneath. This is actually useful. Overlaid fields will normally contain text titles, headings and box characters while input or display fields, possibly with scrollable data, will be placed on top. In fact, two field types are supported called '<background>' and '<text>'. The former has extra properties and would normally fill the entire panel area. Firstly it provides a background colour for the panel. But also it provides various default attributes for other fields relating to the mouse, and also a detailed level of help accessible from any field.

Thus the first defined field should be the '<background>'. Subsequent fields are defined on top of what is already there, but a facility allows re-ordering in case a background field is forgotten. If customised colours are required, this is the best time to use the Colours option, though the facility can be used at any time.

...Back to the Example

Let us now suppose that the main panel of the example in Figure 1 is being edited. The screen would appear as illustrated in Figure 2.

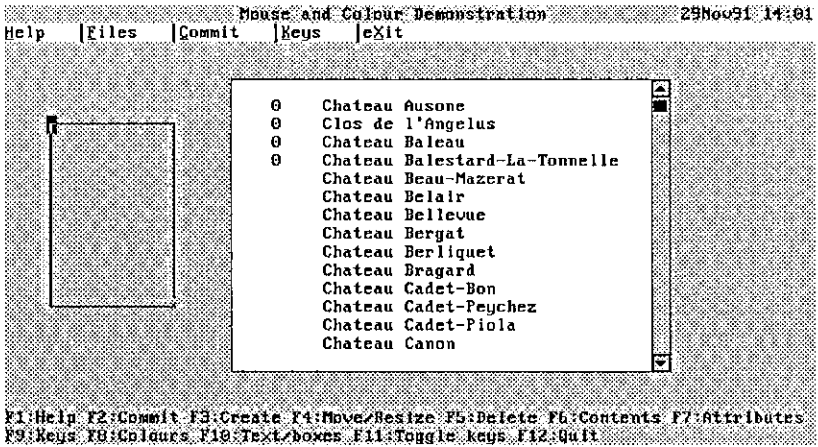


Figure 2: Creating a new field

The individual fields are filled with data if available and take defaults otherwise. The figure indicates that data was available for the title and for the list of wines (as variables in the workspace) but not for the year field. The date/ time is generated by a utility function. Text on the menu bar is built into that field. Note that the function key box relates to the panel design facility, not to the panel being defined.

An alternative display option toggled by the Text/Boxes key shows fields marked by boxes in place of the contents display. This is useful for seeing exactly where each field lies. In the figure, the demarcation between the year, wine list and background cannot be differentiated. By switching to Boxes mode, the boundaries would be clear.

The figure shows a new field being created to the left using the mouse. The blob at the top-left is the initial creation point. The cross at the opposite corner is the mouse, dragging the field to the required size. On release it will become the current field, highlighted by a solid border. The current field is the topmost one in which the cursor lies. Clicking with the mouse moves the cursor to that point, making it the current field. An existing field may also be moved using the Move/Resize option or by dragging with the mouse.

...Field Attributes

Pressing the Attributes key, or double clicking with the mouse on a field, pops the attributes panel up. Figure 3 illustrates this panel shown after double-clicking on the year field. It further shows the field data type, highlighted, in the process of being defined through a second cascading series of pop-up panels.

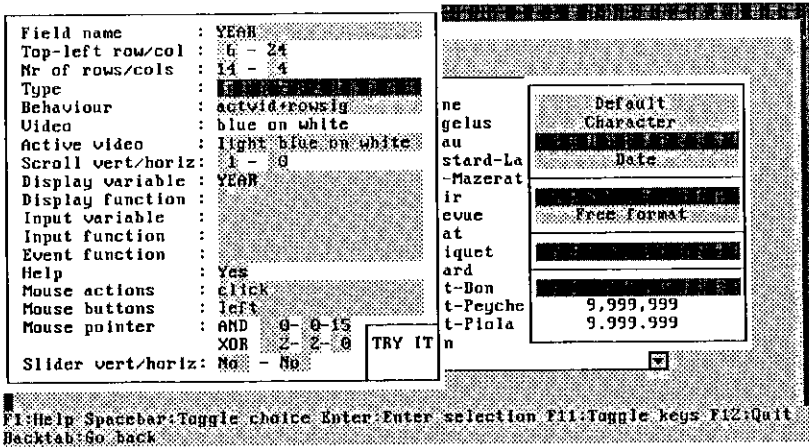


Figure 3: Editing the type attribute

The left-hand column identifies field attributes. The cursor or mouse may be moved to any data area on the right. Several inputs respond to the Enter key or mouse click by displaying a pop-up (as in the example). Others allow direct input.

Taken in order, the attributes are as follows:

Field name. A user-defined name, or Enter/click presents a pop-up of special names. <background> and <text> have already been mentioned. In addition, there are:

- <command> to define a menu bar field;
- <time> to define a date/time field;
- <title> to define a title field;
- <slider-#> defining system-generated slider bar fields.

Some of the remaining attributes may be protected when a special field is chosen.

Row/column. Defines the top-left screen coordinates and the number of rows and columns. These may be changed.

Type. Defines the type of data that will appear in the field. Options are Character, Numeric or Date with finer details such as formats for each. Default allows Character or Numeric, but would generally be chosen for non-input fields. A Character field type with a comprehensive set of edit facilities, including word wrap, is included.

Behaviour. The options here are partly implementation-dependent. It allows various behaviour attributes when the field is entered, such as input protection and clearing on input; if control is to be returned such as on entry, after modification, and on exit; and when the field is left, to retain active video. The required options are selected through a pop-up panel.

Video. Specifies the foreground and background colours for the field in normal state, and optionally when the cursor is in the field. A pop-up panel provides a bar to choose the colours, illustrating what the selection looks like. The example shows the different active video for the Type field.

Scroll. Relates this field to other fields with the same vertical or horizontal scroll number. All fields with the same vertical scroll number will scroll in parallel when any of the fields is scrolled. Ditto for horizontal scroll groups.

Display variable. Normally the name of a variable that will supply the contents of the field. For a protected field, it can be any APL expression yielding an appropriate array.

Display function. An optional function (or APL expression) to be applied to the display variable whose result then supplies the field contents. This is useful for special formatting of, say, numeric data.

Input variable. A variable to receive the result of the field when it is modified by the user. It defaults to the display variable if not defined.

Input function. A function (or APL expression) to alter or verify an item of the field when modified. It might, for instance, check that a numeric value is within a given range, or it could change a response to a nicer format, or remove superfluous blanks.

Event function. This is a function that allows an exit event to be checked to see if it applies. For example, a left or right arrow movement key might be an event to return control to the main program, but special action applies only to certain fields, otherwise the normal movement action is to be taken. This facility allows the latter case to be filtered out, thereby improving performance.

Help. A pop-up edit panel allows help text to be composed for the field. This is displayed when a help key is pressed (normally F1 or the Help menu bar command) with the cursor in the field. If the help key is pressed a second time, the generally more detailed help composed for the <background> field is displayed.

Mouse details. Mouse actions and buttons are selected from pop-up menus. The appearance of the mouse pointer is specified by two sets of character-foreground colour-background colour triplets that are respectively "exclusive ORed" and "ANDed" with whatever is on the screen at the current pointer position. These actions are of course handled by the mouse driver. A small box labelled TRY IT allows you to see what the pointer will look like. Mouse details can be set individually for a field or set globally for unspecified fields.

Sliders. Sliders can be specified both vertically and horizontally. On exiting, sliders will be generated automatically around the field. A slider can also be generated around a set of fields sharing the same scroll group either vertically or horizontally, but not both. If the result is not what is required, the fields generated for the sliders can be edited.

An option is included to save the set of attributes as the defaults. These are inherited by new fields as they are created, and also by unnamed fields.

...Setting Function Keys

The panel to set function keys is actioned by the Keys option from the main panel. It is illustrated in Figure 4.

Key Definitions

Box Keys: ? Yes Short Title :

Std Key	Std Name	User Key	User Description	Mouse Information
1 F1	help	F1	Help	Actions click
2 F2		F2	Commit	Buttons left
3 F12	quit	F12	Quit without saving	Pointer AND 255-15- 0
4 ER				XOR 0- 0- 2
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				

Colours

— Box —

Key: _____

Help text

F1:Help F2:Commit F3:Order F5:Delete F9:Show F11:Toggle keys F12:Quit
Key definitions

Figure 4: Defining function keys

Key details are defined in four parallel columns. The first identifies all keys that will cause control to be returned to the program. This may include any key, not just function keys. The next column indicates the standard use for that key, if it has one, from 'help', 'keys' and 'quit', all of which invoke special action. The third and fourth columns describe the key mnemonic and a brief description of its purpose, used to compose the help block at the base of the screen. These items are left blank for keys that are not to be included in the block. The numbers on the left can be edited to change the presentation order.

The keys may be shown with or without a surrounding box, and in the former case with a short title inserted within the lower box. The colours used and the mouse details may also be chosen. Having compiled the list, the Show option provides a preview of what it will look like. Key actions are not field specific. However, an action can be filtered out using the Event Function attribute described above.

...Saving the Panel

Having completed the panel definition process, the Commit option presents a pop-up panel showing the panel file and panel name, if they have been defined, and a short description. The names may be changed, and a new file or panel name created if necessary. This provides a means of copying a panel between files, and of using an existing panel as the starting point for a new panel. A warning message gives the current status of file and panel.

Finally, software can generate a skeleton control function for the panel, localising all referenced variables, identifying function references and providing a control structure for events implied by the key actions and other events. Figure 5 provides a sample for your perusal.

```

▽ PAN=TEST1:OSM:ERROR:MESSAGE:KEYS:P:RGB:SMROWS:SMUPD:Z:TEXT:TITLE:YEAR
[11]  a* Demonstration panel
[12]  a Control for panel TEST1 generated by program
[13]  a Referenced variables:
[14]  a TEXT TITLE YEAR
[15]  a Referenced functions:
[16]  a* CLEANTEXT
[17]
[18]  TITLE=' '
[19]  TEXT=' '
[10]  YEAR=0
[111] P=P_SETUP'd:\MS\pantest\P_GET'TEST1'          a Get panel and setup
[112] RGB=P_SETCOLOURS P                             a Set colours, return prior
[113] SMROWS=P_P_MAINFIELD'Commands YEAR TEXT'      a Input field numbers
[114] SMUPD='*'                                       a Fields to refresh (all)
[115] Z*(=SMROWS)1 1                                 a Initial context
[116] KEYS='F2' 'F12' 'ER' 'E' 'C' 'X' 'M1' 'M2'
[17]
[10] Read:Z P=P P_MOUSEREAD SMROWS*'Z SMUPD * SMUPD*0
[19]
[20]  +Modified Key[(Z[15]P_BEHAVIOUR modified)\1]
[21]  a Exit events
[22]  Modified:~NotYet
[23]
[24]  Key:~KeyF2 KeyF12 KeyER KeyE KeyC KeyX KeyM1 KeyM2 None[KEYS\Z[11]
[25]  a Key events
[26]  KeyF2:~NotYet
[27]  KeyF12:~NotYet
[28]  KeyER:~NotYet
[29]  a Menu bar events
[30]  KeyE:~NotYet
[31]  KeyC:~NotYet
[32]  KeyX:~NotYet
[33]  a Mouse events
[34]  KeyM1:~NotYet
[35]  KeyM2:~NotYet
[36]
[37]  NotYet:ERRQR*'Event ',(4>Z),' not yet available' * Z[4]+c'' * +Read
[38]  None:ERRQR*'Unknown event ',4>Z * Z[4]+c'' * +Read
[39]
[40]  Exit:SETCOLOURS RGB
▽

```

Figure 5: Sample generated control function

Support Tools

Designing a panel is just the first part of the process. A suite of tools is provided to be used by the programmer for all sorts of tasks. The basic tools deal with setting up and reading from the keyboard and mouse actions. The latter tool is in fact the engine that drives the application. Other tools allow control of the colour palette, hiding of the key block, disabling or highlighting of menu bar items, pro forma selection panels, and many other facilities.

...The Engine Room

The "read" tool warrants a closer look. Its arguments pass through the panel definition variable; fields to visit and in which order when using movement keys (tabbing and back-tabbing); keys to intercept and return control (or the standard set); the current context, i.e. where the cursor and mouse are, and optionally the next action to take; and fields to be refreshed as a result of any action taken by the main program of which the panel software would be unaware.

The tool constantly monitors the keyboard and mouse, taking the appropriate action when any events occur, such as updating the screen, scrolling, displaying help information, toggling keys, validating modified fields, and returning control to the main program. It is only the latter occurrence that requires the main program to take over. The current context and the possibly updated panel definition variable are returned. The current context identifies the cursor and mouse positions, the last key and exit event, or mouse event and button, and flags for updated fields. This context vector may be used, modified and returned to the "read" tool to adjust the environment.

Footnote

I should add that although this is a text-based package written in Dyalog APL, based to some extent on their screen manager, the principles are much more general, applicable to and able to be implemented on most other APL platforms running under DOS.

References

- [Cannon 91] Ray Cannon: *'Mice do it on the Mat'*, Vector 7.3 p.130
- [Smith 90] Adrian Smith: *'Lots More VGA Colours'*, Vector 7.1 p110
- [Duncan 88] Ray Duncan: *'MS-DOS Functions'*, Microsoft Press
- [Norton 88] Peter Norton/Richard Wilton: *'Programmer's Guide to the IBM PC'*, Microsoft Press

J

SPECIAL FEATURE

This section of VECTOR is given over to a set of three articles about J. The Questionnaire surveys Vector readers' opinions on Ken Iverson's new APL dialect; it is followed by some questions and answers which may help the experienced APLer convert to J; then there is Donald McIntyre's tutorial on the application of the J constructs 'hooks' and 'forks', and finally a short piece by Paul Chapman showing how J can be used to explore a simple problem in mathematics.

J Questionnaire	Jonathan Barman	90
J Questions and Answers		
Questions	Jonathan Barman	94
Answers	Roger Hui and Ken Iverson	
Hooks and Forks and the Teaching of Elementary Arithmetic	Donald B. McIntyre	101
Cross-clocks in J	Paul Chapman	124

In future issues of Vector, we will include further articles by Donald McIntyre, on boxed arrays, reading external data, and Jacobi's method for eigenvalues. We will welcome submitted articles about J, and will give them the same consideration as submitted material in traditional APL.

J Questionnaire

by Jonathan Barman

In mid November I sent out a questionnaire to everyone who had had a copy of J to find out what their reactions were to the language. Articles for Vector have to be ready by 30 November, so this left very little time for replies. It was gratifying that 15 people were able to respond by that date.

Several of the respondents have got J but have no knowledge of APL, and their views are particularly interesting as their thinking has not been coloured by APL.

Q: Was J easy to install?

The majority found J so easy to install that they must have wondered why I asked the question. Chas Yates was the only one to have problems. He found that the tutorial files were written for J version 3.0 whereas he had been sent version 2.9. The 'spelling' is different between the two versions, so he had to fiddle about changing such things as : to :: and [to } . He also had to replace the line ending characters. I think Chas was unlucky in this respect, as my version 2.9 of J for the IBM PC had the correct tutorial files. Chas also had to change the J.PRG file to J.TOS to stop the Desktop (Presentation Manager) fouling up the system.

Q: When you first fired up J did you find any difficulty with the editor? Was it obvious how to exit from J?

The majority had no difficulty with the editor or in exiting from J. Donald McIntyre pointed out that the session manager has changed greatly and that the present version in 3.5x4 is great. The APL users found no difficulty with)off to exit. Now that exit has changed to 0! : 55 in version 3.5 it is suggested in STATUS.DOC that you assign off = . 0! : 55 in your PROFILE.JS.

The Atari users did not get an editor at all. Characters can be typed on the command line and there is a destructive backspace, but nothing else. Not surprisingly, the Atari users were pretty rude about the 'editor'. Chas Yates hoped that the Atari was not going to be fobbed off as merely a games machine. It provides a cheap solution for those who cannot afford a PC or a Mac.

Q: Is J easy to learn? Does a knowledge of APL help the learning process?

Most found J difficult to learn, and felt that knowledge of APL was essential given the current state of the documentation. Simon Barker summed it up with the comment that learning J without the aid of a wide range of material means playing with the interpreter and relying on APL skills to aid experimentation. Donald McIntyre said that J is not easy to learn from the Dictionary, and, like APL, is best learned from a teacher.

Those who had little knowledge of APL had a slightly different view. Chas Yates felt that J was no harder than Forth, and Mark Whippey said that it was easy enough to get started. R Gillan said J was enjoyable to learn and useful in some areas of mathematics, especially induction and recursion formulae.

Q: Do you like the Dictionary of J as a definition of the language?

The Dictionary was generally liked. Simon Barker felt that it was a succinct and clear description of the language and useful as a basic reference text.

Q: Could you learn J from the Dictionary of J? Was the tutorial in Appendix B useful?

Donald McIntyre pointed out that it is not intended that you should learn J from the Dictionary. Since the Dictionary was all that Konrad Hinsien had, it had to be sufficient. Another commented that he felt that he was worrying away like a dog with a rag when someone could explain it better. Others felt it would be possible to learn J from the Dictionary, but it would take a lot of effort.

Several respondents with early versions of J did not have Appendix B of the Dictionary, but those who did have it found it useful.

Q: Which features of J are difficult to understand?

All of them, says Donald McIntyre, if you do not have the information! Simon Barker had difficulty using conjunctions and adverbs to derive verbs because they require a very clear understanding of how these constructions work in J. He also felt that J 'spelling' is more difficult than APL because of the use of modifiers to produce related symbols. Chris Brunson and Malcolm Rigg found function definition difficult, and the Suite variable an awkward method of program control.

Q: Would you have found it easier to learn J if there were more extensive examples and exercises?

A resounding YES to this one.

Q: Did you enjoy learning J?

Another resounding YES.

Q: Which features of J are not very useful and could be dropped from the language?

Most dared not comment, mainly because they had limited experience with J. Chris Brunson suggested that either APL-like programs with line numbers should be introduced, or insist entirely on tacit definition and operators for everything.

Q: What additional features should be included in the language?

Dave Ziemann suggested a better front end. Donald McIntyre and Chris Brunson would love graphics support. The remainder felt that they did not know enough about J to comment.

Q: Does J give you new insights into programming techniques? Can you see improvements that should be carried over into APL?

Most felt that they had benefited from studying J. Simon Barker likes the way that scan has been redefined to apply a verb monadically to increasing subsets of data. Chris Brunson likes the ability to assign verbs, adverbs and conjunctions to names, and could see this feature being incorporated in a new version of APL. Donald McIntyre says J is APL!

Q: What should J be used for?

The majority felt that J was best for teaching, education, intellectual stimulation, mathematicians and APL-type problems. Donald McIntyre uses it for everything now. Chris Brunson finds it useful for geographical data since he can handle arrays of complex numbers, but finds the inability to link J to graphics functions frustrating.

The only contrary note was from E Sargeant who thinks J will quickly become shelfware.

Q: What sort of problems would be easier to solve in J than any other language?

Dave Ziemann summed it up by saying that J is best for data transformations and array processing.

Q: Would you be able to use J commercially?

The majority of replies were a definite NO. Chris Brunson thought that there might be a possibility if the 640K barrier was broken, and Konrad Hinsen felt that an up-to-date user interface would be essential.

Q: Why did you get a copy of J?

The main reason seemed to be due to an interest in new languages, particularly one developed by Ken Iverson. Chas Yates and Konrad Hinzen wanted the power of APL without the hassle of special characters.

Q: Do you like J?

Everyone said YES. Chris Brunson thinks that J is an excellent idea, both as a 'tool of thought' and for rapid solutions of certain problems in practice. His complaints are just minor irritations.

One question that I did not ask was about the character set, and it was interesting to note that two users specifically got J because of APL character set problems. Simon Barker had ambivalent feelings about the subject. He likes being able to write and print J scripts in ASCII, but feels that J has lost something valuable without APL's symbology. APL was primarily a powerful notation that was then implemented as a programming language, but J is primarily a programming language that can be used as a (reasonable) notation. Because of this he finds J much harder to read than APL, J text is littered with the period and colon modifiers necessary to provide its rich symbol set and subsequently sacrifices some of the clarity and conciseness of APL.

Many thanks to the users who took the time and trouble to reply to the questionnaire. It is marvellous to be able to share their experiences in the language at this early stage of its development.

J Questions and Answers

Questions by Jonathan Barman

Answers by Roger Hui and Ken Iverson

When preparing for this issue of Vector my first and most essential task was to study J. Some questions occurred to me which I could not answer immediately from the documentation supplied, so I wrote to Roger Hui for elucidation. Roger Hui and Ken Iverson have kindly agreed that their answers may be published. Their reply grouped all their answers under five main topics. I have inserted the questions next to each of their answers.

STATUS.DOC, Version, and Editions

JB: The STATUS.DOC file marks many features as Not Yet Available. When do you plan to make these features available?

RH & KI: The file STATUS.DOC records the history, current state, and machine-dependent aspects of the implementation. Items are marked NYA (not yet available) if they are described in the dictionary but are not yet implemented, and serve to explain nonce or other errors that would be signalled if these items are invoked. Of the more than 180 nouns, verbs, derived verbs, adverbs, conjunctions, and other language facilities described in STATUS.DOC, only thirteen are marked NYA. These items are not available in any other APL dialect either, and their current unavailability does not impair the use of the rest of the system. Thus we would not characterise the situation as one where "many features" are not yet available.

Of the thirteen NYA items in version 3.4, four (infinity and indeterminate) are now partially available and a fifth (polynomial roots) is under active development. These will be released in due course.

JB: Do you expect that J will change in a way which might make it difficult for users to move to a new version? At this early stage in development I suspect that you would want to keep a free hand, and not be constrained by programs written in previous releases of J.

RH & KI: In the 18 months since J was first released, there have been fifteen versions, resulting from releasing new features as soon as they are implemented. We plan to have less frequent but more substantial releases in the future. As the implementation history in STATUS.DOC indicates, there are very few changes which make older versions incompatible. Incompatibilities have been mostly in the form of spelling changes, a form of incompatibility readily dealt with by a text editor. Continuing to use older versions is not the

problem that it is on mainframes; in particular, the size of the J system permits the co-existence of several or all versions in the average personal computer.

JB: I noticed that STATUS.DOC describes features that are not in the Dictionary, for example 9! : 10 and 9! : 11. As the *Dictionary of J* [4] is dated 1991, I assume that you will not issue a revision in the very near future. How often to you expect to issue revised versions of the Dictionary?

RH & KI: New editions of the dictionary are generally produced when the language is significantly extended; over the past 18 months, there have been four editions. From time to time, STATUS.DOC (being more malleable) can be slightly more up-to-date than the printed dictionary. This is the case with 9! : 10 and 9! : 11 (query and set system command names): at the time of printing, the current edition of the dictionary anticipated that system commands would be replaced by new verbs and hence there would be no need for 9! : 10 and 9! : 11. Except for these two items, STATUS.DOC describes no language features not in the dictionary.

Working with J

JB: Editing and debugging J seems to be extremely difficult. What techniques do you advise for editing defined verbs? I would expect to be able to write a verb that does the necessary translation to and from the format suitable for 8! : 9, but could not see how to do it in the short time I have been playing with J.

RH & KI: One might organise the work as follows: Definitions relevant to a particular problem are kept in a script file. (Any explicit definitions are included.) The interactions in a session depend on the machine: on systems supporting windows (such as the Macintosh or UNIX X-windows), the script file would be in a separate text editor window, and executing the script in a J window occurs concurrently with editing the script in the editor window. On single-screen systems, executing the script in J alternates with editing the script (without terminating the J session), using the 8! : 9 editor or PC-WRITE (PC) or the vi editor (UNIX), or whatever editor is available in the underlying system. Here, using the standard ASCII character set is an advantage: standard editors handle J scripts with ease.

Given this way of working with scripts, we seldom attempt to edit the display representation (5! : 2) of objects, preferring instead to edit the source script.

JB: How should one implement Stop and Trace? Is it possible to define a verb or adverb to do the job? Page 69 of *Programming in J* suggests adding a leading L, but this did not seem to produce any output for my defined verbs.

RH & KI: If `pr = . 1 : 2 & 2`, composing `pr` with tacit verbs or prefacing each line of an explicit definition by `pr` provides an execution trace. The use of insertions in programs to get the effect of trace and stop is admittedly awkward, and we intend to provide further facilities.

Formatting

JB: Formatting in J appears to be very limited. I was expecting something along the lines of the APL2 type formatting, rather than the full `□FMT`. Negative numbers are carefully avoided in the illustration of a report on page 53 of *Programming in J*. I had hoped for a demonstration of the techniques for adding decorations to numbers. Do you plan to introduce more powerful formatting, or do you feel that this is not appropriate for J?

RH & KI: It is unlikely that format ("`:`") will be extended to do "picture format" or to insert parentheses, commas, currency symbols, and the like. The current "`:`" is adequate and has no problems with negative numbers. There was no deliberate intent to avoid negative numbers on page 53 of *Programming in J* - that example is a report on return on investment, and may return on investment stay ever positive! More elaborate reports can be written by the user, as the *Programming in J* example illustrates. Alternatively, one could forward arrays to external presentation packages for further processing. The "host" (`0! :`), file (`1! :`), and LinkJ [3] interfaces make this possible and easy.

Line Labels

JB: The example loop shown on page 18 of *Programming in J* has line numbers hard coded, and there is no illustration of the use of line labels. I assume that you just ran out of room for a relatively uninteresting topic, rather than having anything against line labels.

RH & KI: In explicit definition, the sequence control `$.` permits sequencing more elaborate than in other dialects, and the use of line labels with `$.` permits sequencing analogous to the branch arrow. *Programming in J* was first published in October 1990; line labels only became available on 1991 2 15. Hence the absence of examples involving line labels.

Tacit and Explicit Definition

JB: When one of my initial efforts at defining a verb crashed with an index error I was not aware if the verb was still on the execution stack. What happens when an error is encountered? Can one look at local variables? `)s1` gives a nonce error; will this feature be included soon?

RH & KI: Several of your questions concern the explicit definition of verbs (those derived from the `:` conjunction). In our own work, we hardly ever use the explicit form; instead, a typical application consists of a set of tacit definitions [1]. For example:

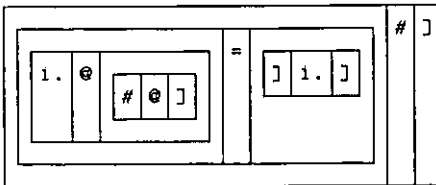
```
mean    =. +/ % #
var     =. mean @ *: @ (- mean)
stddev  =. %: @ var
```

The problem is decomposed into meaningful subunits that are solved by separate, simpler verbs. These verbs can be tested and debugged independently; intermediate results of the overall verb, being just results of the intermediate verbs, are readily available. The decomposition serves as excellent documentation. This style of programming (of course) does not originate with J, and is practicable in other APL dialects. See for example *Programming Style in APL* [2].

JB: As an exercise I rewrote the looping example shown on page 18 of *Programming in J* with line labels, setting a local variable `lines=.L1;L2`, and then selecting the lines to be executed with `$.>(1<.y.){lines`. In an attempt to remove the parentheses in the latter statement I tried to create the tacit form with `'>(1<.x.){y.' : 11` to see if that would give a clue on how to glue the thing together. All I got was an `index error`. The `{` seems to cause the problem, as substituting another verb (e.g. `+`) produces a result. Is this a bug?

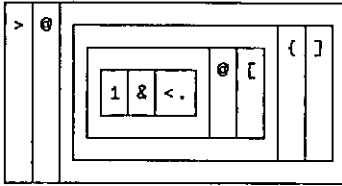
RH & KI: An explicit sentence on nouns `x.` and `y.` resulting in a noun, can be translated into a tacit verb using `: 11`. For example:

```
nub =. '(i.#y.)=y.i.y.)#y.' : 11
nub
```



The translator works by executing the sentence and produces a tacit verb as a by-product. In this execution the system initialises `x.` and `y.` to 1 by default. This is why `'>(1<.x.){y.' : 11` failed: 1{1 should signal index error. The dictionary does provide sufficient information to derive a remedy: In general, the left argument to `: 11` is a boxed list consisting of the sentence to be translated and non-default values of `x.` and `y.`. Thus:

(1; '>(1<.x.){y.';3 4) :11



JB: I have not worked through all the examples in Chapter 5 of Programming in J, but having got the tacit form with `a=. '>(1<.x.)+y.' : 11` and converted it with `b=. 5!:2 <'a'`, I found difficulty replacing the + with {. Having made the replacement it was annoying to discover that 5!:0 is not yet available as an inverse for 5!:2! What should I have done?

RH & KI: We are sorry that you were disappointed that `fix 5!:0` currently does not define an object from its display representation. It is quite possible that 5!:0 will never be so extended, because the display representation is not unique (for example, `'+. ' &`, and `+. &`, have the same display). However, 5!:0 does invert 5!:1 (atomic representation) and 5!:3 (string or WSIS representation), and one could modify a verb by manipulating these representations. We plan to define and implement linear representations (5!:5), which is more convenient for editing. (For example, for nub defined as above, `5!:5<'nub'` would produce `'(i.@{#@]) =]i.] #]'`.) Linear representation makes a useful addition to the 5!: series:

- 5!:0 fix
- 5!:1 atomic (gerundial) representation
- 5!:2 display representation
- 5!:3 string (WSIS) representation
- 5!:4 tree representation
- 5!:5 linear representation NYA

JB: I am not sure I understand exactly when a Hook or Fork is interpreted instead of the normal application of verbs. If I put the verbs in parentheses, or assign them to a proverb, then Hook or Fork is always used. How would I prevent the application of Fork to a train of 3 verbs which is assigned to a proverb? For example, `a +* b` is not a Fork, but assigning the verbs `f=. +* b` means that `a f b` is a Fork. I am nervous that a such a simple rearrangement should change the way in which the verbs are applied. Is this unreasonable?

RH & KI: As stated in the dictionary, a hook or fork is produced by an isolated sequence of verbs; the isolation may be produced by punctuation (parentheses) or be a sentence such as mean = . + / % #. Punctuation may drastically change meaning in any language, as illustrated by the pair of APL expressions (+/x)++/x and +/x++/x, and by the pair of English sentences "The teacher said he was stupid" and "The teacher, said he, was stupid".

ED: *Obviously a silly question! Donald McIntyre explains the topic in some detail in his article "Hooks and Forks and the Teaching of Elementary Arithmetic".*

References

- [1] Hui, R.K.W., Iverson, K.E. and McDonnell, E.E, *Tacit Definition*, APL91 Conference Proceedings, APL Quote-Quad, Volume 21, Number 4, 1991 8.
- [2] Iverson, K.E., *Programming Style in APL*, Proceedings of the APL Users Meeting, I.P. Sharp Associates, 1978 9. Reprinted in *A Source Book in APL*, APL Press 1981.
- [3] LinkJ allows C to call J and J to call user-written C functions.
- [4] Iverson, K.E., *ISI Dictionary of J*, Iverson Software Inc., 1991.

Typesetter's Note

As with APL, J is developing a tradition of using a simple monospaced font (such as Courier) for code and sample output. This has the advantage that things are easy to line up, but a particular problem with Courier (apart from the fact that it is too light on the page) is that the standard PostScript font lacks any of the PC line-drawing characters. This makes a nonsense of most of the examples!

Accordingly, I have had a quick hack at my APL-2741 font, essentially to make it upright (rather than oblique), and slightly heavier. Page 72 shows the two fonts side by side if you want to see the general effect. As it is very likely that Vector will continue to print articles about J, I would like to get the new font as tidy as I can, so that it adds to (rather than detracts from) the readability of the code. Please address any comments to Vector Production, at APL-385.

The font will, of course, be put into the public domain as soon as the design has stabilised.

J

where to find it ...

I-APL is UK agent for
Iverson Software Inc and
holds stocks of J for
PC and Macintosh

Tel: 0727-860130

Hooks and Forks and the Teaching of Elementary Arithmetic

by Donald B. McIntyre

Introduction:

After teaching APL to colleagues and students for many years [1,2,3], in retirement I am discovering the age-levels at which various mathematical concepts can be usefully introduced to children with the aid of the executable notation J. Teachers at Edradour School, Pitlochry, are helping me look at the possible use of J [4-8] to help children learn arithmetic. Although lacking Zdenek Jizba's experience in the elementary classroom [9], I hope the examples given here will encourage teachers to investigate J with their pupils. The underlying notation and syntax of the language are explained for the teacher, who must decide how much to disclose to particular pupils.

Algebra passed through three stages: *rhetorical*, *syncopated*, and *symbolic*. At first words were used, without other symbols. The words were then abbreviated, and eventually the abbreviations became so contracted that the origin of the symbols was forgotten. New symbols were devised for operations hitherto unknown or not formalised [10-12]. J provides a large number of symbols, but the teacher (or user) can name these, thus reverting to the rhetorical (or perhaps syncopated) stage. This may at times be a convenience for any user; in particular a teacher may find that children are at first more comfortable with words than symbols.

As we grow we seem destined to recapitulate the intellectual history of our race. Pupils should nevertheless be encouraged to use symbols as early as possible. Dantzig put it well: "Greek thought was essentially non-algebraic, because it was so concrete. The abstract operations of algebra, which deal with objects that have purposely been stripped of their physical content, could not occur to minds which were so intensely interested in the objects themselves. *The symbol is not a mere formality*; it is the very essence of algebra. Without the symbol the object is a human perception and reflects all the phases under which the human senses grasp it; replaced by a symbol the object becomes a complete abstraction, a mere *operand* subject to certain indicated operations. ... The symbol has a meaning which transcends the object symbolised; that is why *it is not a mere formality*. It is the power of transformation that *lifts algebra above the level of a convenient shorthand*." [11, p.80, 87. Italics in original].

The J dialect:

Mathematics is a language [11-14]. Indeed Hogben, in a book of great popularity and influence, included a chapter on the grammar of mathematical language [13], distinguishing the *verbs* (functions) and *nouns* (data) of mathematical sentences. Hui and Iverson et al have gone further, recognising and implementing *adverbs* (monadic operators) and *conjunctions* (dyadic operators) [15]; Bernecky and Hui have shown the power of *gerunds* (function arrays) for parallel processing [16]; and Iverson and McDonnell added *phrasal forms* and *pronouns* [17]. J uses *pro-verbs*, and similar names assigned to adverbs and conjunctions [4].

J is a powerful dialect of APL available as shareware implemented for a large number of computers [18]. Its roots are in Iverson notation [19] and in APL's method of Direct Definition of functions [20, 21, 12]. The spelling adopted uses ASCII characters, either alone or immediately followed by a period (.) or colon (:). Thus + is *plus*, +. is *or*, and +: is *double*; % is *divide*, %. is *matrix divide*, and %: is *square root*. As in all dialects of APL, both monadic and dyadic meanings are recognised. Assignment is written =. and read "is".

When an expression is executed the result is produced and can be assigned to a variable. If no assignment is made the result is displayed. In the examples below expressions for execution are shown after three spaces at the beginning of the line and the results are shown on the next line and beginning in the first column of the line. If the expression is a verb or a pro-verb (the name of a verb), then the definition of the verb is shown in boxes (see examples later in the article).

Names

Nouns, verbs, adverbs, conjunctions, and gerunds can all be assigned names. If the name *plus* is easier to remember than the symbol +, simply assign and use the name. Names used instead of nouns are pronouns; pro-verbs stand for verbs in a similar way; and other parts of speech can be named also. Here I define some names for use in the rest of this article. The rows of the table are the defining expressions in J and can be entered like this on a computer running J. In J the comment symbol is NB. and anything after that on a line is ignored. Definitions given here are informal; see the *Dictionary* for details [4].

Pro-verbs (or verbs, for short):

add=. +	NB. synonyms can be helpful. See plus
behead=. }.	NB. drop first item
copy=. #	NB. x # y is x copies of y
divided_by=. %	NB. 12%3 is 4

double=.	+	NB. +: 5 is 10
floor=.	<.	NB. <. x drop any fractional part of x
format=.	" :	NB. width and precision of output
halve=.	- :	NB. -: 10 is 5
head=.	{.	NB. take the first item
increment=.	> :	NB. increm is short for increment. Add 1
laminat=.	. , :	NB. join two lists to make a table
larger_of=.	> . :	NB. x >. y choose the larger value
left=.	[NB. return the left argument. Was {}:
lesser_of=.	< . :	NB. x <. y choose the smaller value
less_or_equal=.	< :	NB. 1 2 3<: 2 is 1 1 0
magnitude=.		NB. y is absolute value of y
match=.	- :	NB. arguments identical
minus=.	-	NB. 4-6 is 2
not=.	- . :	NB. converts 0 to 1 and 1 to 0
off=.	0! : 55	NB. return to DOS with "off 0".
one_minus=.	- . :	NB. 1 - y. extends the boolean not
plus=.	+	NB. synonyms can be helpful. See add
power=.	^	NB. 3 ^ 2 is 9
reciprocal=.	%	NB. reciprocal of 2 is 0.5
residue=.		NB. x y remainder on dividing x into y
reverse=.	. :	NB. change a list abcde into edcba
right=.		NB. return the right argument. Was {}:
script=.	0! : 2@< : (<@ [0! : 2 <@])	NB. read/write script files.
		NB. 'output.fil' script 'input.fil' or
		NB. 'output.fil' script '' or
		NB. script 'input.fil'
show=.]	NB. return the right argument
shape=.	\$	NB. length of a list or number of rows
		NB. and columns of a table
signum=.	*	NB. * _5 0 7 is 1 0 1
tally=.	#	NB. number of items
times=.	*	NB. 3*4 is 12
times_pi=.	o.	NB. Multiply by pi. o.1 is pi
transpose=.	};	NB. turn a table (or flat) on its side
tree=.	5! : 4 @ <	NB. display structure of a defined verb;
		NB. tree 'mean'
wholes=.	i.	NB. list integers, starting with 0.
		NB. 1. 10

Pro-adverbs (or adverbs, for short):

cross=.	~	NB. switch arguments
fix=.	f.	NB. fix a verb: "compile" it into J
		NB. symbols
insert=.	/	NB. insert verb between items
scan=.	\	NB. apply to successively longer subsets

Pro-conjunctions (or conjunctions, for short):

atop=.	after=.	@	NB. create a verb by bonding two verbs
rank=.	"		NB. explained in context
with=.	&		NB. bond a noun to a verb

This use of *with* for bonding a noun to a verb is sometimes called *Currying*, after the mathematician Haskell B. Curry [15]. Note that (plus insert) is a derived verb, just as *run quickly* is a new verb created from an old one modified by an adverb. Also (plus insert scan) is yet another verb, derived from the first by two adverbs, as in *run very quickly*. Conjunctions bond nouns or verbs together to form new verbs as in *divide by 4*, or *run and hide*. Thus:

```
quarter=. % with 4
```

and

```
>: @ i.
```

is a new verb that generates wholes (integers) starting with 1.

Plus, Times, Power:

To add up a list of numbers, insert the verb plus between each item:

```
5      1 plus 1 plus 1 plus 1 plus 1
```

The operation is written concisely by defining the new verb total, and using copy to repeat an item:

```
5      total=. plus insert
      total 5 copy 1
```

Obtain partial totals by using scan:

```
      total scan 10 copy 1
1 2 3 4 5 6 7 8 9 10
```

The sum of six twos is:

```
12     2 plus 2 plus 2 plus 2 plus 2 plus 2
12     total 6 copy 2
```

Multiplication was, of course, invented to do this conveniently:

```
12     6 times 2
```

Similarly:

```
64     2 times 2 times 2 times 2 times 2 times 2
64     times insert 6 copy 2
```

Exponentiation (raising a number to a power) was invented to make this easier:

```

        2 power 6
64

```

The order of the arguments makes a difference (6 copies of 2 is not the same as 2 copies of 6); i.e. power is not commutative:

```

        times insert 2 copy 6
36
        6 power 2
36

```

The adverb **cross** interchanges the arguments:

```

        2 copy cross 6
2 2 2 2 2 2
36      2 power~ 6

```

Sum of a list:

```

        show 1=. increm wholes 10
1 2 3 4 5 6 7 8 9 10
55      total increm wholes 10

```

This total can be obtained (as young Gauss knew) by halving the product of the last number and the last number plus 1 [9]:

```

        i plus reverse 1
11 11 11 11 11 11 11 11 11 11
55      halve 10 times 11
5050    total increm wholes 100

```

Noting that:

```

        1 2 3 + 100 99 98
101 101 101
5050    halve 100 times increm 100

```

And

```

        total increm wholes 1000
500500  halve 1000 times increm 1000
500500

```

Hooks [4, 17]:

The expressions `i plus reverse i` and `y times increm y` are examples of a commonly encountered construction. Two verbs (call them `g` and `h`) are applied to the data (call it `y`) in a special way; namely, the result of applying `h` to `y` becomes the right argument of `g`, and `y` appears again as the left argument of `g`. The syntax of J enables this to be written concisely as:

```
(g h) y instead of y g (h y)
```

This construction is called a *hook*. Note that `g`, with arguments both to the left and right, is dyadic, whereas `h`, with an argument on the right only, is monadic. The resulting hook is in this case monadic (the argument appears only once). Thus, instead of writing:

```
      i plus reverse i
11 11 11 11 11 11 11 11 11 11
```

write:

```
      (plus reverse) i
11 11 11 11 11 11 11 11 11 11
```

Similarly:

```
      halve 100 times increm 100
5050      halve (times increm) 100
5050
```

The sum of the positive whole numbers from 1 to `n` is given by the defined verb `spwn`:

```
      spwn=. halve after (times increm)
      spwn 1000
500500
```

Parentheses are needed to get the correct hook, because a conjunction (`after`) seizes the item immediately to its right as its right argument.

Dyadic hooks are also common. They occur when `x` is modified by some function of `y`. The syntax is:

```
x (g h) y instead of x g (h y)
```

Here is an example. How many steps are there if starting at `_2` we go to 5? [22] Not counting the `_2` we start at, there are 7 steps:

```
      # _1 0 1 2 3 4 5
7
```


This number is the magnitude of the difference between `_2` and `5`:

```
7      (magnitude atop (minus insert)) _2 5
```

Or using J symbols:

```
7      n=. | @ (-/)
        n _2 5
```

If we proceed in steps other than unit steps, we must divide by the step size. The required verb is a hook combining the two verbs `into` and `n`. Name it `h`:

```
into=. %~
nsteps=. into n
```

Then for half steps, we have:

```
14     0.5 nsteps _2 5
```

Display `nsteps` to see that it consists of two adjacent verbs; i.e. a hook.

```
nsteps
┌───┬───┐
│ into │ n │
└───┴───┘
```

Fix `nsteps`, so that subsequent changes to its components (`into` and `n`), will not affect it.

```
nsteps=. nsteps fix
nsteps
┌───┬───┬───┐
│ %~ │ |@ │ -/ │
└───┴───┴───┘
```

```
(( %~ ) ( | @ ( -/ ) ))
```

This is a display of the definition of `nsteps`. It shows how the expression will be parsed. There are two outer boxes, and hence this is a hook. Contiguous boxes show that the symbols in them are to be grouped together. We can immediately translate the display into the fully parenthesised form written immediately below it. There is no harm in using the fully parenthesised form, but experimentation will show whether parentheses can be omitted with impunity. In this case the parentheses round `-/` are needed to prevent the conjunction `atop` (`@`) from taking the `-` instead of `(-/)` as its right argument. Adverbs are monadic,

taking the item to the left as the argument. Thus cross (~) modifies the verb `divided_by` (%) to its left and no parentheses are needed to ensure this. So `nsteps` can be written with a single set of parentheses. No spaces are needed but I have used them to draw attention to the two groups of symbols that make the hook. As we shall see, three groups make a fork.

```
nsteps=. %~ |@(-/)
0.5 nsteps _2 5
14
```

Two hooks are used in the scaling (or normalising) of a list of numbers to make the range from 0 to 1. Such scaling is often needed when preparing graphical display.

Let `list` be the name of the list of numbers nine, three, four, negative 2, ..., seven:

```
list=. 9 3 4 _2 12 1 _4 15 7
```

The smallest value will be 0 if we subtract the smallest value. This should be quickly recognised as a monadic hook:

```
list - <./list
13 7 8 2 16 5 0 19 11
(- <./) list
13 7 8 2 16 5 0 19 11
q=. minus lesser_of insert
q=. - <./
q list
13 7 8 2 16 5 0 19 11
```

Similarly:

The largest number will be 1 if we divide `list` by the largest value. This is another monadic hook.

```
p=. % >./
```

`scale` is the composite verb in which `p` is applied to the result of `q`; i.e. `p` is atop `q`

```
scale=. p@q
0.3 format scale list
0.684 0.368 0.421 0.105 0.842 0.263 0.000 1.000 0.579
```

Forks [4, 17]:

A fork is a succession, or train, of three verbs. The meaning assigned is this:

(f g h) y	is	(f y) g (h y)	Monadic
x (f g h) y	is	(x f) g (x h y)	Dyadic

The mean is an example of a monadic *fork*. Compute it by dividing the total [of the list] by the tally [the number of items in the list]:

```
y=. 1 2 3 4
(total y) divided_by (tally y)
2.5
```

Display mean in various ways:

```
mean=. total divided_by tally
mean
```

total	divided_by	tally
-------	------------	-------

The string of three boxes shows that mean is a fork. The same information is given by another kind of display, called a parse tree.

```
tree 'mean'
- mean — [ total
           | divided_by
           | tally
```

Fix the verb, "compiling" it into primitive J symbols.

```
mean=. mean fix
mean
```

+	/	%	#
---	---	---	---

```
tree 'mean'
- mean — [ / — +
           | %
           | #
mean 1 2 3 4
2.5
```

mean is a monadic fork because it takes only a right argument. The syntax is:

```
(f g h) y is the same as (f y) g (h y)
```

The *arithmetic progression vector* (apv) given below is an example of a dyadic fork. This is its syntax:

```
x (f g h) y is the same as (x f y) g (x h y)
```

An example of a fork including a hook as one prong is the verb `clean`, which sets to zero small values resulting from round-off errors. For data take:

```
show y=. 1 10 100 1000 10000 into 1.2345
1.2345 0.12345 0.012345 0.0012345 0.00012345
```

Let the right argument y be the data to be cleaned, and let tolerance be the left argument x; then the result is y times the truth or falsity (1 or 0) of the proposition that the tolerance is less-than-or-equal to the magnitude of y:

```
y * (x proposition y)
```

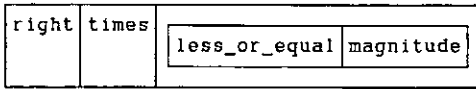
The proposition is a dyadic hook:

```
0.01 (less_or_equal magnitude) y
1 1 1 0 0
0.01 <: (| y)
1 1 1 0 0
```

times is, of course, the central term of the fork:

The verb clean is therefore:

```
clean=. right times (less_or_equal magnitude)
clean
```



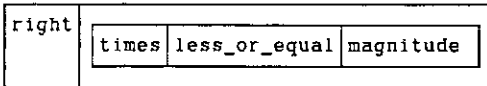
```
tree 'clean'
```



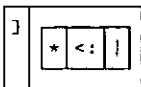
```
x=. 0.01
y=. 1 10 100 1000 10000 into 1.2345
x clean y
1.2345 0.12345 0.012345 0 0
x clean -y
_1.2345 _0.12345 _0.012345 0 0
```

If we left off the parentheses then less_or_equal would be the central term of a fork, which along with right would make a hook.

```
f=. right times less_or_equal magnitude
```



```
f=. f fix
f
```



This would be quite wrong! Consider why this is so. A dyadic hook like this can be expanded:

$x (g h) y$ expands to $x g (h y)$

In our case:

```

      g=. J
      h=. *<:|
      x (g h) y
1 0 0 0 0
      x g (h y)
1 0 0 0 0

```

Because g is the dyadic verb `right` it returns its right argument; so x can make no contribution to the result, which must depend solely on the monadic function h applied to y .

```

      h y
1 0 0 0 0

```

Defining the following pro-verbs:

```

p=. *
q=. <:
r=. |

```

Expand the fork h :

```

      (p q r) y
1 0 0 0 0
      (p y) q (r y)
1 0 0 0 0

```

But

```

      r y
1.2345 0.12345 0.012345 0.0012345 0.00012345

```

And

```

      p y
1 1 1 1 1
      (* y) <: (|y)
1 0 0 0 0

```

The monadic times $(*)$ is called the `signum`. The result is `1`, `0` or `_1` depending upon whether the argument is positive, zero or negative. Hence with this definition of f

```

f=. right times less_or_equal magnitude

```

the left argument is ignored and the result is 1 if the `signum` of `y` is less than or equal to the absolute value of `y`. This is true unless `y` is a positive number lying between 0 and 1. This might be a useful function in another situation, but it is not what we want for `clean`!

Iverson's utility verbs `by` and `over` provide a convenient way to illustrate the result. See the Appendix for an explanation.

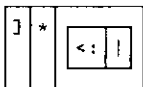
```
x=. 9 0.5 0 _0.5 _9
y=. _9 _1.5 _1 _0.5 0 1 e_3 0.3 0.4 0.999 1 2 10 100
over=. ((,.,.y;.)y):y,
by=. ' '2;@,.,@[,.]
x by y over x f"0 1 y
```

	_9	_1.5	_1	_0.5	0	0.001	0.3	0.4	0.999	1	2	10	100
9	1	1	1	1	1	0	0	0	0	1	1	1	1
0.5	1	1	1	1	1	0	0	0	0	1	1	1	1
0	1	1	1	1	1	0	0	0	0	1	1	1	1
_0.5	1	1	1	1	1	0	0	0	0	1	1	1	1
_9	1	1	1	1	1	0	0	0	0	1	1	1	1

The rank conjunction (") here instructs `f` to use rank-0 cells (atoms, units or scalars) from the left argument with rank-1 cells (lists, longs or vectors) of the right argument.

We can return to the definition of `clean` with added understanding. It is helpful to use spaces in order to draw attention to the three verbs of the fork, but spaces are not required:

```
clean=. ] * (<:|)
clean
```



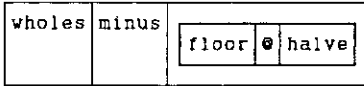
The Number Line:

Negative numbers once seemed *absurd* or *fictitious* [23, p.252], but a number line makes it easy for a child to grasp a concept that once taxed the greatest mathematicians. In a few years of instruction we are each led through the stages that took our ancestors generations to achieve. Experiments should help:

```
show i=. wholes 10
0 1 2 3 4 5 6 7 8 9
  10 plus i
10 11 12 13 14 15 16 17 18 19
```

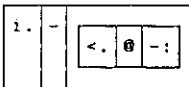
Number lines are produced by the following fork:

```
n1=. wholes minus floor atop halve
n1
```



Or, more concisely in J symbols:

```
n1=. 1. - <.@-:
```



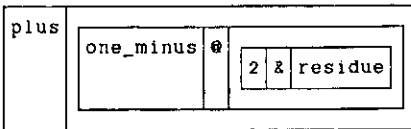
```
      n1 11
_5 _4 _3 _2 _1 0 1 2 3 4 5
```

Various number lines can be produced simultaneously by using the rank conjunction (") to specify that rank-0 cells (atoms, units or scalars) on the left are to be combined with rank-1 cells (lists, longs, or vectors) on the right:

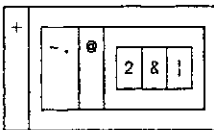
```
      1 10 100 1000 times"0 1 n1 11
_5  _4  _3  _2  _1 0  1  2  3  4  5
_50  _40  _30  _20  _10 0  10  20  30  40  50
_500  _400  _300  _200  _100 0  100  200  300  400  500
_5000  _4000  _3000  _2000  _1000 0  1000  2000  3000  4000  5000
```

The argument for n1 should be odd. To ensure that an even argument is made odd, define odd as a monadic hook:

```
odd=. plus one_minus@(2&residue)
odd
```



```
odd=. + -.@(2&|)
odd
```



```
      odd 2 4 6 8 laminate 1 3 5 7
3 5 7 9
1 3 5 7
```

This works by dividing the argument by 2 and subtracting the remainder (residue) from 1. The result is then added to the argument. Because the remainder on division by 2 is either 0 or 1 (depending on whether the number is even or odd) subtracting it from 1 changes a 0 to a 1 and a 1 to a zero. A list (vector) of 0s and 1s is called *logical* (taking 1 as true and 0 as false) or *boolean* (after George Boole, who taught us how much can be done with an algebra restricted to these two numbers). In logic the verb not converts true to false and false to true. Boolean algebra is, however, part of algebra, and its 0 and 1 can be combined with ordinary numbers by algebraic operations. Because there is no difference between the verbs one_minus and not (other than their domain), J uses the same symbol (-.) for both.

Define the verb nline, which applies n1 after using odd:

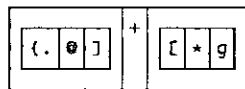
```
nline=. n1 @ odd
nline"0 right 12 13
_6 _5 _4 _3 _2 _1 0 1 2 3 4 5 6
_6 _5 _4 _3 _2 _1 0 1 2 3 4 5 6
```

Using the verb nsteps (defined above to determine the number of steps) produce the required number of wholes (integers):

```
g=. 1.@>.@>:@nsteps
0.5 g _2 5
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
0.5 g _2 5.2
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

The *arithmetic progression vector* is a fork containing a fork that in turn contains a hook. The outside fork is: the first item of the right argument ($t.\theta$) plus the fork $[*g$, which in turn is the left argument t times the dyadic application of g .

```
apv=. {.@] + [*g
apv
```



```
0.5 apv _2 5
_2 _1.5 _1 _0.5 0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5
```

Using J primitives (or fixing apv with fix):

```
apv=. {.@] + [ * 1.@>.@>:@(%~ |@-/)
```


Place value:

In learning arithmetic one of the first tasks is to understand place notation; i.e. the values assigned to digits at each successive place in a number. With this in mind we explore as follows:

```
times insert scan 10 10 10 10 10 10
10 100 1000 10000 100000 1000000
```

Because successive multiplications are produced by power:

```
10 power 1 2 3 4 5 6
10 100 1000 10000 100000 1000000
```

Looking at this result a pupil might wonder what the result of 10 power 0 would be:

```
10 power 0 1 2 3 4 5 6
1 10 100 1000 10000 100000 1000000
```

which leads in turn to the exploration of negative values:

```
10 power _4 _3 _2 _1 0 1 2 3 4 5 6
0.0001 0.001 0.01 0.1 1 10 100 1000 10000 100000 1000000
```

The reciprocals are place values:

```
reciprocal 10 power _4 _3 _2 _1 0 1 2 3 4 5 6
10000 1000 100 10 1 0.1 0.01 0.001 0.0001 1e_5 1e_6
```

The value assigned to a place is given by the verb place:

```
place=.10 with power
place 3
1000
```

Because we write numbers so that the larger place values are to the left (ancient Egyptians usually did the opposite), the values of successive places are:

```
reverse place wholes 7
1000000 100000 10000 1000 100 10 1
```

Continuing the series to the right:

```
|. 10^ 1 apv _4 4
10000 1000 100 10 1 0.1 0.01 0.001 0.0001
```

Rounding:

Having grasped place notation, pupils are ready for rounding. This must be done mentally, but at an appropriate stage the formal process should be introduced.

Because J is an executable mathematical notation, its use has the advantage that computing skills are taught at the same time.

```

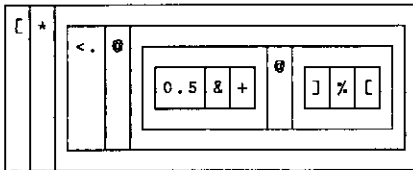
400    100 * <.0.5+438%100    NB. Round 438 to the nearest 100
30     10 * <.0.5+26%10      NB. Round 26 to the nearest 10
12.3   0.1 * <.0.5+12.345%0.1 NB. Round 12.345 to nearest 0.1
12.4   0.2 * <.0.5+12.345%0.2 NB. Round 12.345 to nearest 0.2
    
```

Rounding will, in general, be to an integer multiple of a given number. This is defined formally as:

```
round=. ' ' : 'x. * <. 0.5+ y.%x.'
```

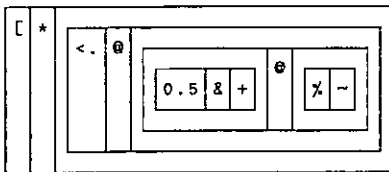
where x. and y. are place-holders for the left and right arguments respectively. The empty string preceding the conjunction (:) means that we are defining the dyadic case only. Because the arguments are referred to explicitly, this is an example of *explicit* definition, as opposed to the *tacit* definitions used exclusively above. It can be translated to tacit form by the adverb :11 [24]:

```
'x. * <. 0.5+ y.%x.' : 11
```



This is a fork with * as the central verb. Because the fork]%[can be written %~ we have:

```
round=. [ * <. 0.5% + (%~) )
round
```



```

10 100 1000 round 146464
146460 146500 146000
y=. 2 4 6 8
show x=. y round 146463
146464 146464 146466 146464
    
```

Verify that these results are indeed (integer) multiples of y

```

x%y
73232 36616 24411 18308
y=.3 7 11 15 25 33 125
show x=. y round 146464
146463 146461 146465 146460 146475 146454 146500
x%y
48821 20923 13315 9764 5859 4438 1172
    
```

Rounding 438 to the nearest 100

```

(10^2) * <.0.5+438%10^2
400
    
```

Round to given number of decimal places

```

rdp=. ' ' : '(10^x.) %~ <. 0.5 + y. * 10^x.'
0 1 2 3 4 5 rdp 0.1
3 3.1 3.14 3.142 3.1416 3.14159
    
```

Rewriting in tacit form:

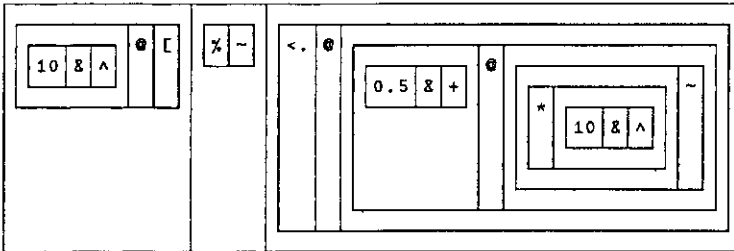
```

rdp=. 10&^@[ %~ <.@(0.5&+@[ * 10&^@[ ])
    
```

This contains two forks, but the last one ([* 10&^@[]) can be written as a hook ((* 10&^@)-), though this necessitates additional parentheses.

```

rdp=. 10&^@[ %~ <.@(0.5&+@(( * 10&^@)-))
0 1 2 3 4 5 rdp 0.1
3 3.1 3.14 3.142 3.1416 3.14159
rdp
    
```



Rounding to a given number of decimal places is, of course, only a special case of round:

```
(10^-1.6) round 0.1
3 3.1 3.14 3.142 3.1416 3.14159
```

The hook is obvious in the following:

```
h=. (]*<.0(0.5&+0%)) 10&^
rnd=. h-
0 _1 _2 _3 _4 _5 rnd 0.1
3 3.1 3.14 3.142 3.1416 3.14159
x=. 1.5
y=. 146464 14646 1464 146
table=. (10^x) round"0 1 y
```

Notice that we simultaneously round many numbers to several different places and produce a table of rounded values. (Iverson's utility verbs by and over are explained in the appendix).

y by x over transpose table

	0	1	2	3	4	5
146464	146464	146460	146500	146000	150000	100000
14646	14646	14650	14600	15000	10000	0
1464	1464	1460	1500	1000	0	0
146	146	150	100	0	0	0

```
table match x rnd "0 1 y
1
```

Letting f be the fork and h the hook:

```
f=. ] * <.0(0.5&+0%)
h=. (f 10&^)-
table -: x h"0 1 y
1
```

Examples of rounding exercises from a school workbook [25, p.14, 15, 22]:

```
n. : 10 round n=.12 26 165 14 38 43 56 65 97 145 235
12 26 165 14 38 43 56 65 97 145 235
10 30 170 10 40 40 60 70 100 150 240
n. : 100 round n=.170 438 650 160 250 463 729 607 896 717 91
332 548
170 438 650 160 250 463 729 607 896 717 91 332 548
200 400 700 200 300 500 700 600 900 700 100 300 500
10 100 1000 round 8478
8480 8500 8000
```

Round the numbers n to the place values given by p:

```

n=. 8217 4096 7358 6105 8654 5583 7950 6008
p=. 10 100 1000
table=.j: p round"0 1 n
n by p over table

```

	10	100	1000
8217	8220	8200	8000
4096	4100	4100	4000
7358	7360	7400	7000
6105	6110	6100	6000
8654	8650	8700	9000
5583	5580	5600	6000
7950	7950	8000	8000
6008	6010	6000	6000

Concluding Remarks:

This paper illustrates some aspects of the J dialect of APL using examples related to the teaching of elementary arithmetic. It demonstrates that hooks and forks are ubiquitous, and shows how to use them in reading and writing J. Examples are given of displays that show how expressions are parsed. These are invaluable aids in understanding the language.

Definitions are given in *tacit* form. They are *functional* as advocated by Backus [26].

Because J is evolving, it is essential to note which version is used. Examples are included of changes that were made between versions. The files `status.doc` and `xenos.doc`, provided with the system, document changes from earlier versions. All examples included in this paper have been executed with Version 3.5x4.

As from Version 3.5x1, 1991 8 26, the three remaining "system commands" have been replaced by verbs created by the external conjunction (`!:`).

```

)script    replaced by 0!:2
)sscript   replaced by 0!:3
)off       replaced by 0!:55

```

It is convenient to include the following verbs in the file `profile.js`, which is loaded automatically at the start of a session.

```

script=. 0!:2@< : (<@[ 0!:2 <@])
off=. 0!:55

```

`script` is an ambi-valent verb that permits specification of output and input script files; e.g. `'output.fil' script 'input.fil'` Because `off` is a verb, it must have an argument. To terminate a session and return to DOS, enter `off 0`.

Remembering Klein's *Elementary Mathematics from an Advanced Standpoint* [27], in this paper I have treated topics in elementary arithmetic from a relatively advanced viewpoint. I hope teachers will find stimulation for themselves and be able to select examples suitable for pupils at various levels. Even very young children can use some of the examples, if only to check their answers. The exercise could provide an excellent introduction to both computing and mathematics. The literature of J is still limited. This paper may be useful as an introduction for anyone wishing to try J.

Acknowledgements:

Kenneth Iverson, Roger Hui, and Eugene McDonnell (Iverson Software Inc.) have provided generous support on numerous occasions. Mrs. Rosemary Roache and Julian Romanes (Edradour School) gave insight into the teaching of elementary mathematics. Anthony Camacho and Graham Woyka encouraged me to write about learning J.

Donald B. McIntyre
Luachmhor, Church Road, Kinfauns, Perth PH2 7LD
Scotland - U.K. Telephone: 0738-86-726

Appendix: Displaying a table with over and by:

```
over=. {{. .@;}.)@":@,
by=. ' ' &;@,.@[. ]
```

Notice that `over` contains the fork `{. .@; }`. and `by` is built upon the fork `' ' &;@,.@[.]`

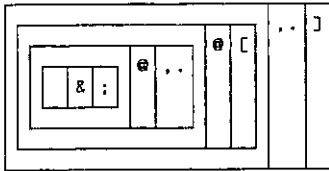
While reference [8] was in press, the symbols for *Ravel Items* and *Raze* were interchanged (Version 3.2, June 1991). Because the monadic form *Ravel Items* changed from `"`; in J3 to the form `" . "` (as in J3.5x4, 1991 10 22), the versions of `over` and `by` given in [8] must now be written as follows:

```
over=. . . @ ({. ; }.) @ ": @ ,
by=. ( , ~" _1 ' ' &; @ , . ) ~
```

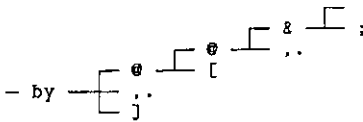
In these versions, `over` contains the fork `{. ; }`. and `by` is built on the hook `(, ~" _1) (' ' &;@ , .)`

J's displays are invaluable aids in learning to read and write verbs like these. Compare two versions of by each in two types of display:

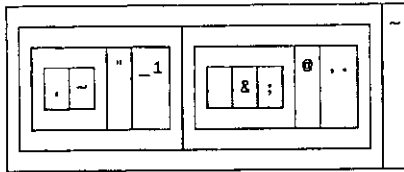
```
by=. ' '&;@.@[.]
by
```



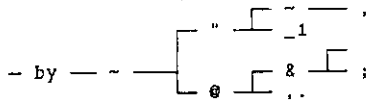
tree 'by'



```
by=. (." _1 ' '&; @ .)~
by
```



tree 'by'



References

- [1] Donald B. McIntyre, *Introduction to the Study of Data Matrices*, In: Models of Geologic Processes: an Introduction to Mathematical Geology. Short Course Lecture Notes, Philadelphia, November 1969. Edited by Peter Fenner. American Geological Institute, Washington D.C. (1969) p.A1-A44, B1-B17, C1-C4.
- [2] Donald B. McIntyre, *The Architectural Elegance of Crystals made clear by APL*, In: Conference Proceedings, APL Users Meeting, Toronto, September 1978. Sponsored by I.P. Sharp Associates, Toronto (1978) p.233-250.
- [3] Donald B. McIntyre, *APL in a Liberal Arts College*, In: Conference Proceedings, APL Users Meeting, Toronto, October 1980. Sponsored by I.P. Sharp Associates, Toronto (1980), p.544-574.
- [4] Kenneth E. Iverson, *ISI Dictionary of J*, Version 3.3 with Tutorials, Iverson Software inc., Toronto (1991) 32pp.
- [5] Kenneth E. Iverson, *Programming in J*, Iverson Software inc., Toronto (1991) 71pp.
- [6] Kenneth E. Iverson, *Tangible Math*, Iverson Software inc., Toronto (1991) 33pp.
- [7] Kenneth E. Iverson, *A Personal View of APL*, IBM Systems Journal, Vol. 30, Number 4 (1991) In Press.
- [8] Donald B. McIntyre, *Mastering J*, APL91 Conference Proceedings, Stanford, California, August 1991. APL Quote Quad Vol. 21 Number 4 (August 1991), p.264-273.
- [9] Zdenek V. Jizba, *Science Education in California*, Vector vol.8 number 2 (1991) p.22-24.
- [10] Florian Cajori, *A History of Mathematical Notations*, The Open Court Publishing Company, La Salle, Illinois. Volume 1 (1951) 451pp. First published 1928; Volume 2 (1952) 367pp. First published 1929.
- [11] Tobias Dantzig, *Number: the Language of Science*, London, George Allen and Unwin, Ltd. 2nd edition (1942) 320pp. First Published 1936, 4th edition 1962.
- [12] Donald B. McIntyre, *Language as an Intellectual Tool: From hieroglyphics to APL*, IBM Systems Journal, Vol. 30, Number 4 (1991) In Press.
- [13] Lancelot Hogben, *Mathematics for the Million*, London, George Allen and Unwin, Ltd (1936) 678pp.
- [14] Kenneth E. Iverson, *Notation as a Tool of Thought*, 1979 Turing Award Paper, Communications of the A.C.M., Vol.23, Number 8 (August, 1980) p.444-465

- [15] Roger K.W. Hui, Kenneth E. Iverson, E.E. McDonnell, and Arthur T. Whitney, *APL\?*, APL90 Conference Proceedings, Copenhagen, Denmark, August 1990. APL Quote Quad Vol. 20, Number 4 (July 1990) p.192-200.
- [16] Robert Bernecky and Roger K.W. Hui, *Gerunds and Representations*, APL91 Conference Proceedings, Stanford, California, August 1991. APL Quote Quad Vol. 21, Number 4 (August 1991) p.39-46.
- [17] Kenneth E. Iverson, and E.E. McDonnell, *Phrasal Forms*, APL89 Conference Proceedings, New York City, August 1989. QuoteQuad, Volume 19, Number 4, (1989) p.197- 199.
- [18] J is available from Iverson Software Inc., 33 Major Street, Toronto, Ontario, Canada M5S 2K9. Phone (416) 925-6096; Fax (416) 488-7559.
- [19] Kenneth E. Iverson, *A Programming Language*, John Wiley and Sons, Inc., New York (1962) 286pp.
- [20] Kenneth E. Iverson, *Elementary Analysis*, APL Press, Swarthmore, Pennsylvania (1976) 219pp.
- [21] Donald B. McIntyre, *Experience with Direct Definition One-liners in Writing APL Applications*, Conference Proceedings, APL Users Meeting, Toronto, September 1978. Sponsored by I.P. Sharp Associates, Toronto (1978) p.281-297.
- [22] William K. Clifford, *The Common Sense of the Exact Sciences*, Edited with a preface by Karl Pearson. Newly edited and with an introduction by James R. Newman. Preface by Bertrand Russell. With bibliography of W.K. Clifford. Alfred A. Knopf, New York (1946), Sigma Books, Ltd., London (1947) 249pp.. Reprinted by Dover Publications, Inc., New York (1955). Chapter 1, Number, Section 11, Steps.
- [23] Morris Kline, *Mathematical Thought from Ancient to Modern Times*, Oxford University Press, New York (1972) 1238pp.
- [24] Roger K.W. Hui, Kenneth E. Iverson, Eugene E. McDonnell, *Tacit Definition*, APL91 Conference Proceedings, Stanford, California, August 1991. APL Quote Quad Vol. 21 Number 4 (August 1991), p.264-273.
- [25] Roy Hollands, *Ginn Mathematics Level 4*, Teachers' Resource Book, Ginn and Company Ltd, Aylesbury, Bucks. (1983), Third impression 1985, 160pp. Intended for 8 to 9 year old children.
- [26] John Backus, *Can programming be liberated from the Von Neumann style? A functional style and its algebra of programs*. 1977 Turing Award Lecture. Communications of the ACM, Vol. 21, number 8 (1978) p.613-641.
- [27] Felix Klein, *Elementary Mathematics from an Advanced Standpoint*, Third Edition first published 1924-1925. English translation published by Dover Publications Inc., New York. 2 Volumes, (1939) 274pp. and 214pp.

Cross-clocks in J

by Paul Chapman

This article was prepared using J version 3.4a running on an Archimedes 540.

Here is an interesting problem whose solution was required to be implemented in C as part of Midland Montagu's Windowed Technical Trading Environment (WITTE). In the absence of a good theoretical understanding of modulo arithmetic and diophantine equations, I chose to explore the problem using J.

A 'clock' ticks with period p , a whole number of seconds, and first ticks a seconds after some agreed original time t_0 , where a is called the alignment. The problem is this: given two clocks, $cx=(px, ax)$ and $cy=(py, ay)$, is there a cross-clock $cz=(pz, az)$, which ticks when and only when both cx and cy tick, and if so what are pz and az ?

Before we begin, let's define two verbs and two adverbs, all very useful in tacit definitions:

```
x. =. [
y. =. ]
X: =. @[
Y: =. @]
```

Now let's describe the data structure for a clock. It is simply a two-element vector of the period and the alignment. In J, we can describe structures by defining functions which extract the members of those structures:

```
p =. {.
a =. {:

C1 =. 5 3
p C1
5
a C1
3
```

In much of the work that follows, we are going to want to extract the period and alignment of a clock in a dyadic function, so let's define some dyadic functions which extract these values.

```
px =. p X:
py =. p Y:
ax =. a X:
```

```

ay =. a Y:
C2 =. 4 1
C1 px C2
5
C1 ay C2
1

```

The exploration begins: here is a function which shows the times of the first x ticks of a clock:

```

expand =. ay + py * i. X:

] E1 =. 10 expand C1
3 8 13 18 23 28 33 38 43 48

] E2 =. 10 expand C2
1 5 9 13 17 21 25 29 33 37

```

Let's proceed empirically for now. Given the expansion of two clocks, what is the expansion of the cross-clock? It is the intersection of the expansion vectors.

```

cap =. e. # x.
E1 cap E2
13 33

```

From this, we can see that the cross-clock's alignment is 13, and its period is $33 - 13 = 20$. Let's write a function which extracts these values, i.e. which does the opposite of expand:

```

first =. {.
second =. 1&{
contract =. (second - first) , first

contract E1 cap E2
20 13

```

Putting all this together, we can build an empirical solution with an adverb. Its argument is the number of ticks of each clock to be used:

```

Cross =. 'contract@(x.&expand X: cap x.&expand Y:)' : 1
I: =. [.]
Cross =. (&expand) (I: (X: ('cap'I:) Y:)) \ (contract@)
5 3 (10 Cross) 4 1
20 13

```

Now let's look at some other examples, to see if there's a pattern:

```

cross =. 100 Cross
      9 0 cross 7 0
63 0
      2 0 cross 2 0
2 0
      4 2 cross 1 0
4 2
      6 5 cross 9 2
18 11
      10 expand 6 5
5 11 17 23 29 35 41 47 53 59
      10 expand 9 2
2 11 20 29 38 47 56 65 74 83
      10 expand 18 11
11 29 47 65 83 101 119 137 155 173
      6 5 cross 9 3
index error

```

The following conclusions can be drawn:

- if ax and ay are both 0, the cross-clock alignment is also 0;
- the cross-clock of two identical clocks is also identical to them;
- 1 0 is an identity clock;
- the cross-clock period is the lowest common multiple of px and py ;
- some pairs of clocks do not have a cross-clock (the index error above).

My mathematics is sadly no longer good enough to be able to provide proofs of all of the above without quite a struggle, but this is after all an informal exploration: you and I will both have to put some faith in the remnants of mathematical intuition left to me after years of C programming.

Now provided we choose a large enough expansion size, we have a solution to the problem. But it is impractical, since the size of the expansions is going to increase roughly with the size of the periods, and my application required a fast solution for arbitrarily large periods.

One part of the problem is already solved: the cross-clock period is the LCM of px and py . So let's at least rewrite our solution to take advantage of this knowledge:

```

lcm =. *.
Crossalign =. 'first@(x.&expand X: cap x.&expand Y:)' : 1
Crossalign =. (&expand) (I: (X: ('cap'I:) Y:)) \ (first@)

```

```

pz =. px lcm py
az =. 100 Crossalign
cross =. pz , az

6 5 cross 9 2
18 11

```

We can simplify the problem by temporarily moving the time origin to one of the alignments, say ay , and then shifting it back once we've calculated the cross-clock:

```

0 2 + (6 5 - 0 2) cross (9 2 - 0 2)
18 11
translate =. (-&0 2) :. (+&0 2)
6 5 cross&.translate 9 2
18 11

```

So now we only have to address the problem where ay is 0. Another simplification is to change the granularity of time to be the greatest common divisor of px and py :

```

0 2 + 3 * ((6 5 - 0 2) % 3) cross ((9 2 - 0 2) % 3)
18 11
scale =. %&3
6 5 cross&.scale&.translate 9 2
18 11

```

This suggests that the cross-clock exists if and only if the difference between the alignments is divisible by the greatest common divisor of the periods. I am confident that this is true, but I am not equipped to attempt a proof.

Let's generalise these last two improvements:

```

gcd =. +.
f =. px gcd py
cross =. pz , ay+f * ((px , ax-ay) % f) az ((py , 0:) % f)
6 5 cross 9 2
18 11

```

We can't use $\&$. here because there is no way in J to use tacit definition to express its right argument as a function of the derived function's arguments. This is a more complicated formula, but now two things can be guaranteed about the arguments to az : ay is zero, and px and py are coprime (i.e. their GCD is 1).

Let's have a look at how az changes with ax for a couple of period pairs obeying these rules:

```

    5 0 az 3 0
0
    5 1 az 3 0
6
    5 2 az 3 0
12
    5 , "0 i. 5
5 0
5 1
5 2
5 3
5 4
    clocksofperiod =. , "0 i.
    clocksofperiod 5
5 0
5 1
5 2
5 3
5 4
    (clocksofperiod 5) az"1/ 3 0
0 6 12 3 9
    try =. clocksofperiod X: az"1/ y. , 0:
    5 try 3
0 6 12 3 9
15 | 6 * i.5
0 6 12 3 9
    7 try 9
0 36 9 45 18 54 27
    63 | 36 * i. 7
0 36 9 45 18 54 27
    7 4 az 9 0
18
    63 | 36 * 4
18

```

az seems to vary proportionally to ax, modulo pz. So we are nearly there: all we have to do is calculate the constant of proportionality, k, for a pair of periods, which is also (by definition) az where $ax = 1$ and $ay = 0$.

Once again, let's first write down an empirical solution for k so that we can experiment:

```

    k =. (px , 1:) az f. (py , 0:)
    az =. pz | k * ax
    5 try 3
0 6 12 3 9

```

```

7 try 9
0 36 9 45 18 54 27

```

The cross-clock for a pair of clocks ticks when and only when both those clocks tick. So at time k , all three clocks must tick. So k is determined as follows:

```

1: = px | k
0: = py | k

```

So k is a multiple of py , say $k = py * r$, which when divided by px leaves a remainder of 1:

```

1: = px | py * r

```

r is therefore the reciprocal of py , modulo px , which we shall write as $px \text{ mr } py$. This always exists provided px and py are coprime, which we have already guaranteed. $x \text{ mr } y$ is defined such that $1 = x | y * x \text{ mr } y$. We can write an empirical definition of mr as follows:

```

AsScalar =. (.@
mr =. ((1: = x. | y. * i. X:) # i. X:) AsScalar
7 mr 9
4
7 | 9 * 7 mr 9
1

testmr =. x. | y. * x. mr y.
2 3 4 5 6 (testmr"0) 5 7 9 11 13
1 1 1 1 1

```

So now we have a solution for k in terms of mr :

```

k =. py * px mr py
5 try 3 0
6 12 3 9
7 try 9
0 36 9 45 18 54 27

```

Finally, the definition of mr is still unsuitable for implementation in C. A variant of Euclid's algorithm can be employed to implement mr recursively, as follows:

```

mr =. (x. | (1: - x. * y. mr y. | x.) % y.) ' 1: @. (x.=0:)
2 3 4 5 6 (testmr"0) 5 7 9 11 13
1 1 1 1 1

```

It is at this point that I state, with some inevitability, that the proof is left as an exercise for the reader.

Since GCD is also omitted from most C libraries, we have Euclid's algorithm for that, too:

```
gcd =. (<. gcd <. | >.) ' >. @. (<. = 0:) "0
```

Or rather more efficiently:

```
OrderArgs =. 'x.~ ' x. @. y.' : 2
OrderArgs =. (~ ') @. ].
gcd0 =. (y. gcd0 y. | x.) ' x. @. (y. = 0:)
gcd =. gcd0 OrderArgs > "0
```

And we can write LCM in terms of GCD:

```
lcm =. (x. * y.) % gcd
lcm =. * % gcd
```

Checking against the J primitives:

```
all =. *./@.
all (gcd -: +.)/~ >:1.10
1
all (lcm -: *)/~ >:1.10
1
```

Finally, let's build up our general solution, putting back the scaling and translation:

```
pz =. px lcm py
f =. px gcd py
k =. (py % f) * (px % f) mr (py % f)
az =. ay + f * pz | k * (ax - ay) % f
az =. pz | ay + k * ax - ay
cross =. (pz , az) "1

10 expand 10 3
3 13 23 33 43 53 63 73 83 93
10 expand 25 18
18 43 68 93 118 143 168 193 218 243
10 3 cross 25 18
50 43
10 expand 50 43
43 93 143 193 243 293 343 393 443 493
```

Here are two more exercises for the reader: how is it that this still works when $ax < ay$, and why does `cross` return a result even when f doesn't divide $ax - ay$ (and what does the result mean)?

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.

Contents

Hackers' Corner	Adrian Smith	132
Technical Correspondence	De Kerf and Spunde	136
Arrays with Style	Adrian Smith	140

Hackers' Corner: More about VGA Colours

by *Adrian Smith*

As Dave Crossley rightly points out, the VGA is grossly under-exploited by the vast majority of PC software. The irritating thing is that the required code is easy to build, and will work under pretty well any APL you care to name. The only snag is having to spend the best part of £30 on Ray Duncan's book! Accordingly, I thought it might be helpful to compile a little workspace of helpful functions (available from APL-385 free if you send me a disk and enclose return postage, but why not just type them in, and have fun writing a better palette editor?!).

So, with thanks to Dave Selby (for pointing out how easy this is), and Duncan Pearson (for lending me Duncan for the weekend) ... here's all you ever wanted to know about `INT 16` ...

```
VGAΔRESET          A Sets everything back to default
VGAΔBLINK 0|1      A Highlighted | blinking background
VGAΔQPALETTE reg  A Checks current palette setting(s)
VGAΔQCOLOUR clr   A Checks current colour value(s)
```

```

▽ VGAΔRESET;RG;W;C
[1] A Reset VGA video to MODE 3 and restore active screen
[2] W←WGET 3 o C←CURSOR
[3] A AH = 0hex ... BIOS service 0
[4] A AL = 3hex ... standard VGA text mode
[5] RG←3 INT 16
[6] A Reset background ...
[7] RG←4099 0 INT 16
[8] A Put things back as they were ...
[9] WPUT W o CURSOR←C
▽
▽ VGAΔBLINK SW;RG
[1] A Flip Background from blinking to intensified.
[2] A AH = 10hex ... BIOS service 10 (EGA/VGA only)
[3] A AL = 3hex ... subservice 3
[4] A BL = 0/1 ... 0 = intensify; 1 = blink
[5] RG←(4099,SW>0)INT 16
▽
▽ R←VGAΔQPALETTE VEC;AX;RG;lab;CT
[1] A Query VGA palette register(s) VEC
[2] A AH = 10hex ... BIOS service 10 (EGA/VGA only)
[3] A AL = 7hex ... subservice 7
[4] A BH = 0 ... colour returned here
[5] A BL = 0-16 ... palette register to query
[6] AX←(256×16)+7 o R←10 o VEC←VEC
[7] Lp←lab+1+(pVEC)pLp,End,CT+1
[8] RG←(AX,VEC[CT])INT 16
[9] R←R, [RG[2]+256
[10] End←lab[CT+CT+1]
▽
```

```

V R+VGAAQCOLOUR VEC;AX;RG;CT;lab
[1]  A Query VGA colour(s) VEC as RGB values
[2]  A AH = 10hex ... BIOS service 10 (EGA/VGA only)
[3]  A AL = 15hex ... subservice 15h = 21(dec)
[4]  A BL = 0-53 ... colour to query
[5]  A Returns 4-col array of [colour,r,g,b] values
[6]  AX+(256*16)+21 o VEC+,VEC o R+q(4,pVEC)pVEC
[7]  Lp:+lab+1+((pVEC)pLp),End,CT+1
[8]  RG+(AX,VEC[CT])[]INT 16
[9]  R[CT; 3 2 4]+3+, 256 256 rRG[3 4] A returned as G,R,B
[10] End:+lab[CT+CT+1]
V

```

For example, to see the current VGA palette, and the standard RGB colours:

```

VGAARESET o STD+VGAAQCOLOUR VGAAPALETTE 0,115
STD
0 0 0 0 ... 56 21 21 21
1 0 0 42 ... 57 21 21 63
2 0 42 0 ... 58 21 63 21
3 0 42 42 ... 59 21 63 63
4 42 0 0 ... 60 63 21 21
5 42 0 42 ... 61 63 21 63
20 42 21 0 ... 62 63 63 21
7 42 42 42 ... 63 63 63 63

```

The next group of two functions allows you to change things! Obviously, there is scope here to put together a simple palette editor ... mine uses RGB to crank up the colour components and rgb to crank them down. It is also handy to provide + to increase all three colours together, and - to wind them all down.

```

VGAAPALETTE pal,cir A Set Palette (0..15) to cir (0..63)
VGAACOLOUR cir,r,g,b A Set cir(s) (0..63) to red,green,blue

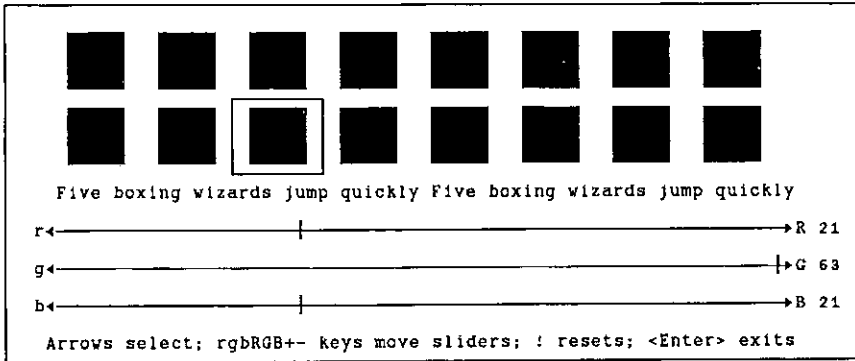
```

```

V VGAAPALETTE VEC;RG;BX
[1]  A Set Vga palette register VEC[1] to VEC[2]
[2]  A AH = 10hex ... BIOS service 10 (EGA/VGA only)
[3]  A AL = 0hex ... subservice 0
[4]  A BH = 0-63 ... required colour
[5]  A BL = 0-16 ... palette register to set
[6]  BX+(256*VEC[2])+VEC[1]
[7]  RG+(4096,BX)[]INT 16
V
V VGAACOLOUR MAT;RG;BX;CX;DX;lab;CT
[1]  A Set Vga colour register MAT[;1] to MAT[;2 3 4] (RGB)
[2]  A AH = 10hex ... BIOS service 16 (EGA/VGA only)
[3]  A AL = 10hex ... subservice 16
[4]  A BX = 0-63 ... colour register
[5]  A CH = 0-63 ... green intensity
[6]  A CL = 0-63 ... blue intensity
[7]  A DH = 0-63 ... red intensity
[8]  MAT+(-2+ 1 1 ,pMAT)pMAT o MAT+((1+pMAT),4)+MAT
[9]  Lp:+lab+1+((1+pMAT)pLp),End,CT+1
[10] BX+MAT[CT;1]
[11] DX+256*MAT[CT;2]
[12] CX+(256*MAT[CT;3])+MAT[CT;4]
[13] RG+(4112,BX,CX,DX)[]INT 16
[14] End:+lab[CT+CT+1]
V

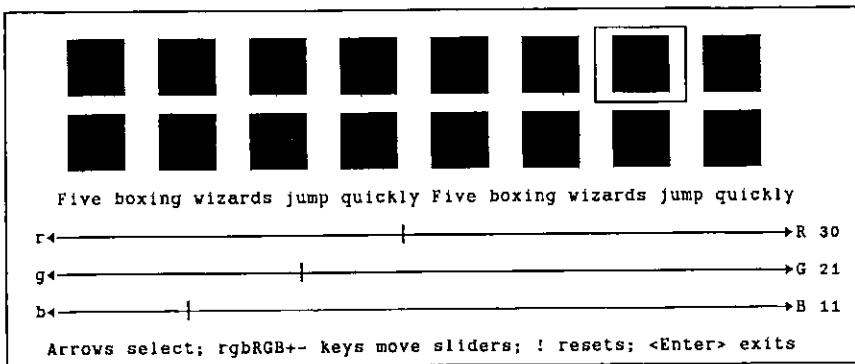
```


As you can see, most of the heavyweight code is to do with making the screen look pretty; if you just want the bare essentials, lines [38] to the end are all you really need! Here are a couple of screen snaps to show the general effect:



This is the standard VGA bright green ... note that it has a barely detectable amount of red and blue added in. Incidentally, the *Five Boxing Wizards* is used to show the effect of all the foreground colours on the selected background, and vice versa. It is a slightly more convenient form of the *Quick Brown Fox*, being only 32 letters long!

For comparison, here is a nice rich brown, which I think is a much better background colour than the standard offering:



That's all folks!

TECHNICAL CORRESPONDENCE

A Note on CPU Time Monitoring

From: Joseph L.F. De Kerf

25 November 1991

In a paper published in Vector [1], Ray Cannon convincingly demonstrates the advantages of the availability of the CPU time monitoring function in APL. Unfortunately, this paper gives the unintended impression CPU time monitoring is a new facility only made available in STSC's APL*PLUS System Function $\square MF$.

In fact, the facility was already made available in the seventies, by Control Data CDC with the system function $\square LTIME$ [2], and by Burroughs, with the system functions $\square SM$ - $\square RM$ - $\square QM$, the results of the monitor being made available through the system functions $\square MC$ and/or $\square MV$ [3]. Later on the facility was made available in VAX APL and Dyalog APL through the system function $\square MONITOR$ and in APL*PLUS and SHARP APL through the system functions $\square FM$ and/or $\square MF$. An overview is given in the accompanying table, the year giving the approximate date of the implementation of the facility.

The most user-friendly implementation seems to me to be that based on the system function $\square MONITOR$. In VAX APL V-2/3/4, the dyadic form of $\square MONITOR$ sets the monitor for the unlocked defined functions/operators specified by the right argument vector or matrix, each row specifying one operation name. The left argument specifies the lines of the operations on which the monitor is to be set. If the left argument contains a zero, the monitor is set to the entire operation. The monitor may be disabled by specifying an empty left argument. The explicit result is a boolean vector, specifying whether the monitor has been set. The monadic form of $\square MONITOR$ returns an n by 3 matrix, where n is the number of monitored lines. The columns give the monitored line numbers, the execution count, and the cumulative CPU time in milliseconds.

In Dyalog APL V-5 and DOS/UNIX, the $\square MONITOR$ behaves about the same, the right argument of the dyadic form however being restricted to the specification of one operation and the explicit result being a vector specifying the line numbers on which the monitor has been set. On the other hand, the monadic

form returns an n by 5 matrix, the columns giving the monitored line numbers, the execution count, the cumulative CPU time, and the elapsed time, both in milliseconds. The fifth column is provisionally reserved and gives zeros.

An example of the use of the `MONITOR` with VAX APL V-2/3/4 is given below. The defined function `FIB N` returns as explicit result the `N` first terms of the Fibonacci Series. Calculation is based on the recursive generator $F(1) = 1$, $F(2) = 1$, and $F(N) = F(N-2) + F(N-1)$. `N` is set to 100.

<i>Bull-Honeywell</i>	<i>APL 64</i>	1980	<code>SM RM MV</code>
<i>Bull-Honeywell</i>	<i>APL 7</i>	1985	<code>SM RM RV</code>
<i>Burroughs/Unisys</i>	<i>APL/700</i>	1974	<code>SM RM QM MC MV</code>
<i>Control Data</i>	<i>APL 2.0 CN</i>	1976	<code>LTIME</code>
<i>Control Data</i>	<i>NOS/VE APL</i>	1984	<code>LTIME</code>
<i>Digital Equipment</i>	<i>VAX APL V2</i>	1985	<code>MONITOR</code>
<i>Digital Equipment</i>	<i>VAX APL V3</i>	1987	<code>MONITOR</code>
<i>Dyadic systems</i>	<i>Dyalog APL</i>	1986	<code>MONITOR</code>
<i>General Electric</i>	<i>OS4000 APL</i>	1979	<code>SM RM MC</code>
<i>Hewlett-Packard</i>	<i>APL\3000</i>	1976	<code>SM RM QM MV</code>
<i>IPSA (Reuter)</i>	<i>Sharp APL</i>	1987	<code>FM</code>
<i>IPSA (Reuter)</i>	<i>Sharp SAX</i>	1988	<code>MF</code>
<i>SLIGOS/BARTS</i>	<i>APL-NET</i>	1978	<code>SM RM QM</code>
<i>STSC</i>	<i>APL*PLUS</i>	1985	<code>MF</code>

```

VR←FIB N
[1] →(N≤2)/0,R←(2|N)ρ1
[2] END:→(N>ρR←R,+/-2↑R)/END
▽
0 1 2 MONITOR'FIB'
1
R←FIB 100
MONITOR'FIB'
0 1 2050
1 1 20
2 98 1940

```

But once again - there is a proliferation of implementing new system commands and system variables/functions. A list of about three hundred system commands and a list of about eight hundred system variables/functions, supported by current implementations, have been published recently [4][5]. Most of the facilities made available, e.g. monitoring CPU time, prove to be valuable enhancements. Unfortunately, the same facility is often implemented in most divergent ways, leading to an inconvenient plethora of synonyms and homonyms. And even if the same distinguished name is used for the same facility, this does not mean that syntax and/or output are the same.

The situation does not only confuse the declared user, but also doesn't inspire confidence on behalf of the potential newcomer, and is a serious obstacle for the future of our language APL. ISO APL and the draft ISO APL Extended only specify a small set of system variables/functions and do not resolve the problems quoted. The main objective of a standard is to promote portability and as such should support those facilities which are common practice. Everybody knows all these things, but "something has to be done on it!".

Joseph De Kerf
 Agfa-Gevaert NV
 AD Informatics
 Septestraat 27
 B-2640 Mortsel

References

- [1] R. Cannon; *How STSC's \square MF can help in testing APL workspaces*; Vector, Vol. 8, No. 2, October 1991, pp. 135-136.
- [2] R. Mayforth; *APLUM-APL at the University of Massachusetts*; APL Quote Quad, Vol. 6, No. 1, Spring 1975, pp. 15-21.
- [3] L. Ryan; *Application Diagnostic Aids in APL/700*; APL 76 Conference Proceedings, Ottawa, Canada, 22-24 September 1976; Edited by G.T. Hunter; Association for Computing Machinery, New York, 1976; pp. 343-345.
- [4] J.L.F. De Kerf; *System Commands and APL - A Survey*; APL-CAM Journal, Vol. 13, No. 2, 16 April 1991, pp. 449-455.
- [5] J.L.F. De Kerf; *APL System Variables and System Functions - A Survey*; APL-CAM Journal, Vol. 13, No. 3, 10 July 1991, pp 744-766.

Horner's Method

From: Walter G Spunde

30 Sept 91

Norman Thomson's neat recursive formulation for calculating Stirling numbers (Vector Vol.8 No.1 p.95) has been gratefully added to my collection of teaching examples. Here is another one which readers of Education Vector may find useful:

Horner's method for calculating values of a polynomial with coefficients $C_0, C_1, C_2, C_3, \dots, C_n$ in ascending order, at points V , is:

$$C_0 + V \times C_1 + V \times C_2 + V \times C_3 + \dots + V \times C_N$$

Texts on numerical methods present this as the most efficient and accurate way of calculating the value of polynomials.

The exercise is to implement these operations, and one solution is:

```
Z ← VS HORNER CS
+0 IF 1 = ρZ + CS
Z ← CS[1] + VS × VS HORNER 1+CS
```

which shows the nesting of linear functions that is involved here, and may be compared with the encoding

$$Z + (\phi ((\rho CS), \rho VS) \rho VS) \perp \phi CS$$

or other algorithms for calculating polynomial values

Walter G Spunde
 School of Information Technology
 University of South Queensland
 AUSTRALIA

DELPHI Panel Design Toolkit & Application Manager			
Improved	Productivity	Benefits	And Value
<ul style="list-style-type: none"> <input checked="" type="checkbox"/> Text-based panel design and control <input checked="" type="checkbox"/> Screen, keyboard and mouse support <input checked="" type="checkbox"/> Customised colours from VGA palette <input checked="" type="checkbox"/> Function library manager <input checked="" type="checkbox"/> Application run-time manager 		<ul style="list-style-type: none"> Modular Performance Price Style 	<ul style="list-style-type: none"> Introductory Combined Price £195.00
<ul style="list-style-type: none"> Available for ... <input checked="" type="radio"/> Dyalog v6 for 386/486 <input type="radio"/> Other APLs - enquire 		<ul style="list-style-type: none"> £190.00 Design Toolkit £90.00 Manager 	

Contact David Crossley (see Vector Product Directory) on 0367 710384

Arrays with Style

by Adrian Smith

Introduction

This is by way of a response to Richard Nabavi's thought provoking piece on array formatting with real (i.e. proportional) typefaces. I don't pretend for a moment that I can offer him a complete solution, but I think he should start by buying two outstanding pieces of current software (Microsoft Word for DOS and Microsoft Excel) and understanding the concept of style sheets.

Incidentally, this could lead to the implementation in APL of the *STYLE ERROR*, first proposed (not entirely seriously) by Robert Bittlestone somewhere in Vector Vol.1.

What is a Style Sheet?

Let's start with MS Word, mainly because I know it better than Excel, and because I can illustrate what I mean as I type. Word attaches something called a style sheet to every document; this defines the appearance of text for:

- each section of the document (of course there may be only one section, even for a long document like VECTOR). This covers things like page-layout (single/double column, margins), running heads, page numbering and so on.
- each paragraph. A paragraph is defined simply as anything between two carriage-returns. The paragraph style defines indents, line spacing, a default character font and size, and various special attributes like 'keep together'.
- each character. Characters normally inherit the style of their parent paragraph, but can be changed individually if required. Oddly, Word has no concept of a *word* as such; it is simply a collection of contiguous characters.

Let's have a look at a selection of paragraph styles by way of illustration:

6	T	Paragraph Heading level 1	centred title - top of page
		Palatino (roman k) 24/28. Centered, space after 1 li (keep in one column, keep with following paragraph).	
7	N	Paragraph 8	Author's name in headings
		Palatino (roman k) 15/18 Italic. Centered, space after 2 li.	
8	S	Paragraph Heading level 2	subheading
		Palatino (roman k) 15/18 Bold. Flush left, space before 1 li, space after 0.5 li (keep with following paragraph).	
9	P	Paragraph Standard	std para
		Palatino (roman k) 13/16. Justified, space after 1 li.	

ADOBE-PL.STY

The style code (T, P etc.) is shown in the left margin of the document, and you 'apply' a style simply by hitting alt+style, e.g. if I were to hit alt+S now, I would get this paragraph in 'subheading' style. If you scan down the first page of this note, you will spot the 'T' and 'N' styles at the top, and also the use of an 'I2' indented block in a slightly smaller type size on reduced spacing. Typical character styles are used to set things like **Bold**, *italic* and (of course) *APL*:

45	B	Character 2	Palatino (roman k) 13 Bold.	boldface
46	IT	Character 3	Palatino (roman k) 13 Italic.	Italic
47	A	Character 4	Courier (modern a) 14 Italic.	APL chars are Courier-Obliq.
48	J	Character 7	Courier (modern a) 13 Bold.	J is Courier Bold
49	Z	Character 5	ZapfDingbats (symbol e) 13.	Dingbats
50	E	Character 6	Palatino (roman k) 8 Bold Superscript.	Exponentials (superscripts)

ADOBE-PL.STY

To set things like x^2 , I simply put the cursor over the '2' and hit alt+E. Of course, you can format text 'directly' in any font you like, but one of the things you soon learn is that this is a **very bad idea!** The whole point about formatting with styles is that the appearance of a document is independent of the content. If I want to switch Vector to New Century Schoolbook, all I have to do is define a new style-sheet (obviously being totally consistent in my use of style names) and attach it to the document. The same would apply to a switch from the current A5 format to (say) double-column A4. As long as everything has been set with style-codes, changes like this are trivial. If the formatting is all convolved with the text (see any Bindweed plant for the meaning of the word *convolved*) as it typically is in WordStar and WordPurfect, life is much harder.

What has this to do with APL?

In many ways, the Excel model is closer to what we need in APL; here you apply a style to a cell (or a group of cells), with a special style (called 'Normal') applied by default to the whole spreadsheet. As well as the character font and size, the cell style specifies things like formatting (£9.99 etc.), colours and borders.

How about this for an idea:

- let's start with a Workspace Style, which specifies the default type style, and includes basic formatting instructions currently scattered about in $\square PP$, $\square PW$ etc. Let's call this style 'Normal' and have it apply to everything we display. It should include a default 'cell width' (in inches, cm or points) for display of numeric scalars or character vectors. Numbers get right aligned in the cell; text is left aligned.

- let's allow the user to edit a style sheet (similar to the examples above) to define a consistent set of display attributes for text and numbers. Styles have simple mnemonic names, and are maintained by APL as a file. Style sheets can be loaded and saved (just like workspaces), and can be attached to workspaces with a command such as:

```

□STYLE+ 'POSTSCRIP.STY'  ⍝ Pick up an existing style sheet
□STYLE
POSTSCRIP
□STYLE+ 'JUNK.STY'      ⍝ This does not exist
2+2
STYLE ERROR            ⍝ Default output fails

```

- now, we need an extra slot in APL's array descriptor: as well as attributes like type, rank and shape, an array also has style (in reality simply a pointer to an entry in the current style sheet). To set the style of our data, we need something that looks and feels a lot like reshape, but instead of modifying the shape elements of the descriptor, it modifies the style:

```

DATA←2 6ρ112          ⍝ Here is an array
DATA←'BOXED'▽DATA    ⍝ Attach the 'boxed' style
DATA

```

1.0	2.0	3.0	4.0	5.0	6.0
7.0	8.0	9.0	10.0	11.0	12.0

```

'currency' ▽ 15
£0.00 £1.00 £2.00 £3.00 £4.00 £5.00    ⍝ These are numbers!

```

... and so on. With nested arrays, you have all the tools you need, as long as you are willing to enclose your data at as low a level as necessary to make each piece of formatting specific to a single array:

```

TXT←'The cat sat on ' ('italic'▽'the ') 'mat'
ρTXT
3
  TXT
The cat sat on the mat
  (▽TXT) (ρ▽TXT)
normal italic normal 3

```

I don't suppose for a moment that this would work as simply as my examples suggest, but it would be fascinating to see some prototypes. Over to you Richard (and John and Dave and Paul); perhaps it will turn up in J for the Archimedes??

Index to Advertisers

APL People (half)	15
Cocking and Drury	6
Delphi (half)	139
MicroAPL	2

All queries regarding advertising in VECTOR should be made to Alison Chatterton, at the address on the inside back cover.

Submitting Material to Vector

The Vector working group meets towards the end of the month in which Vector appears; we review material for issue n+1 and discuss themes for issues n+2 onwards. Please send the text of submitted articles (with diskette as appropriate) to the Editor:

Jonathan Barman,
Hill Top House,
East Garston,
NEWBURY, Berks RG16 7HD
Tel: 048839-575

(not after 10.00pm please!)

Camera-ready artwork (e.g. advertisements) and diskettes of 'standard' material (e.g. sustaining members news) should be sent to Vector Production, c/o Adrian Smith, Brook House, Gilling East, YORK Tel: 04393-385 (6.00pm - midnight).

Product Guide updates should continue to go to Alison Chatterton, as should requests for advertising space.

BAA: Membership Application Form

Membership of the British APL Association is open to anyone interested in APL. The membership year runs from 1st May to 30th April.

Name: _____
 Address Line 1: _____
 Address Line 2: _____
 Address Line 3: _____
 Post or zip code: _____
 Country: _____
 Telephone Number: _____

Membership category (please tick box): 91/92

UK private membership	£12	<input type="checkbox"/>
Overseas private membership	£20	<input type="checkbox"/>
Airmail supplement (not needed for Europe)	£8	<input type="checkbox"/>
Corporate membership	£100	<input type="checkbox"/>
Corporate membership overseas	£155	<input type="checkbox"/>
Sustaining membership	£430	<input type="checkbox"/>
Non-voting student membership (UK only)	£6	<input type="checkbox"/>

I authorise you to debit my Visa/Mastercard account

Number: _____ Expiry date: ____|____

for the membership category indicated above,

- annually, at the prevailing rate, until further notice
 one year's subscription only

(please tick the required option above)

Signature: _____

PAYMENT

Payment should be enclosed with membership applications in the form of a UK Sterling cheque to "The British APL Association", or you may quote your Access or Visa number. Send the completed form (no stamp required from the UK) to:

British APL Association, FREEPOST (SG923), 9 Oak Grove, HERTFORD, SG13 8BR

The British APL Association

The British APL Association is a Specialist Group of the British Computer Society. It is administered by a Committee of officers who are elected by a postal ballot of Association members prior to 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.

1991/92 Committee

Chairman:	David Eastwood 071-922 8866	MicroAPL Ltd., South Bank Technopark, 90 London Road, LONDON SE1 6LN
Secretary:	Anthony Camacho 0727-860130	2 Blenheim Rd, ST ALBANS, Herts AL1 4NR.
Treasurer:	Nicholas Small 081-980 7870	8 Cardigan Road, LONDON E3 5HU
Journal Editor:	Jonathan Barman, 048839-575	Hill Top House, East Garston, NEWBURY, Berks RG16 7HD
Activities:	Dr Peter Branson 081-848 8989	Electronic Data Systems, Stockley Park, UXBRIDGE, Middx UB11 1BQ
Education:	Dr Alan Mayer 0792-295296	European Business Management School, Swansea University, Singleton Park, SWANSEA SA2 8PP
Technical:	Peter Donnelly 0256-811125	Dyadic Systems Ltd., Riverside View, Basing Road, Old Basing, BASINGSTOKE, Hants RG24 0AL
Projects:	John Searle 081-858 6811	4 Hawks Mcws, Greenwich, LONDON SE10 8RA
Publicity:	Misha Jovanovic 0753-853141	99 Oxford Road, WINDSOR, Berks SL4 5DX
Recruitment:	Jill Moss 0225-462602	APL People Ltd, The Old Malthouse, Clarence St., BATH, Avon BA1 5NS
Administration:	Rowena Small 081-980 7870	8 Cardigan Road, LONDON E3 5HU

Journal Working Group

Editor:	Jonathan Barman	048839-575
Secretary:	Anthony Camacho	0727-860130
Production:	Adrian & Gill Smith	Brook House, Gilling East, YORK (04393-385)
Advertising:	Gill Smith	Brook House, Gilling East, YORK (04393-385)
Support Team:	John Searle (081-858 6811), Ray Cannon (0252-874697), Sylvia Camacho, Bridget Barman, Gill Smith	

Typeset by APL-385 with MS Word 5.0 and GoScript

Printed in England by Short-Run Press Ltd, Exeter

VECTOR

VECTOR is the quarterly Journal of the British APL Association and is distributed to Association members in the UK and overseas. The British APL Association is a Specialist Group of the British Computer Society. APL stands for 'A Programming Language' - an interactive computer language noted for its elegance, conciseness and fast development speed. It is supported on most mainframe, mini and micro computers.

SUSTAINING MEMBERS

The Committee of the British APL Association wish to acknowledge the generous financial support of the following Association Sustaining Members. In many cases these organisations also provide manpower and administrative assistance to the Association at their own cost.

APL People
The Old Malthouse
Clarence St, BATH, BA1 6NS
Tel: 0225-462602
Fax: 0225-444552

Compass R&D Ltd
15 Frederick Sanger Rd
Surrey Research Park
GUILDFORD, Surrey GU2 5YD
Tel: 0483-302249
Fax: 0483-302279

Dyadic Systems Ltd
Riverside View, Basing Road,
Old Basing, BASINGSTOKE,
Hants. RG24 0AL
Tel: 0256-811125
Fax: 0256-811130

HMW Trading Systems Ltd
Hamilton House,
1 Temple Avenue,
LONDON EC4Y 0HA
Tel: 071-359 4212
Fax: 071-359 3325

Cocking & Drury Ltd
180 Tottenham Court Rd
LONDON, W1P 9LE
Tel: 071-436 9431

MicroAPL Ltd
South Bank Technopark
90 London Road
LONDON SE1 6LN
Tel: 071-922 8866

Reuters Ltd
1-4 Singer St
LONDON EC2A 4BQ
Tel: 071-867 1166
Fax: 071-867 9792

Peter Cyrix Systems
12 Gloucester Place
LONDON W1H 3AW
Tel: 071-935 2933

Impetus Ltd
Rusper, Sandy Lane
Ivy Hatch, SEVENOAKS
Kent TN15 0PD
Tel: 0732-885126



The British Computer Society, 19 Mansfield Street, London W1M 0BD.