VECTOR

100+ pages of APL News & Views

Including ...

The Journal of the British APL Association

A Specialist Group of the British Computer Society

ISSN 0955-1433

Vol.12 No.2 October 1995

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 (b/w or colour prints welcome). 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, in workspaces from I-APL, APL*PLUS, IBM APL2/PC or Dyalog APL/W, or in documents from Windows Write (use the Vector TrueType font, available free from Vector Production), and Winword-2.

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 1995-96

Category	Fee	Vectors	Passes
UK Private	£12	1	1
Overseas Private	£14	1	1
(Supplement for Airmail, not needed for Europe)	£4		
UK Corporate Membership	£100	10	5
Overseas Corporate	£135	10	
Sustaining	£430	10	5
Non-voting Member (Student, OAP, unemployed)	£6	1	1

The membership year normally 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, Mastercard or JCB, 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 format (A4 or A5) with a 20mm blank border after reduction. Illustrations should be photographs (b/w or colour prints) or line drawings. Rates (excl VAT) are £250 per full page, £125 for half-page or less (there is a £75 surcharge per page 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, Brook House, Gilling East, YORK YO6 4JJ. Tel: 01439-788385 CompuServe: 100331,644

Contents

Editorial: Is There Anybody Out There?

by Anthony Camacho

Does anybody read the notices we send out inserted in Vector?

APLers who organise meetings, write articles or edit magazines need to know what you think and what you want. After a while we will have commissioned all the articles we can think of that we really want to read and organised all the meetings with the speakers we most want to hear. When that time arrives we don't know what to do next unless you tell us.

When we ask you and not one person suggests a speaker or subject (there was one offer to speak!) we get worried. What would you do?

What Sort of Conference?

One prominent member of the APL 96 committee has suggested to me that the traditional APL conference is dead and that what is needed is something much more organised than a tidy arrangement into streams of the papers that fall through the letterbox onto the doormat. "What we need is workshops."

Is it true that the number of papers submitted for review is steadily declining? Is it true that the number of delegates paid for by their employers is steadily declining? Is it true that the importance of APL skills is declining relative to those skills needed to control the operating system? Should APL conferences contain Windows workshops? (Once I thought I was going to spend the rest of my life learning a new word processor every six months; now I think I may have to learn a new operating system every year.)

Who can provide the information needed to enable conference planners to do a good job? All the conferences have been run by SigAPL so they ought to be able to do this.

It would also be nice to know, for each conference, how many exhibitors there have been (and how much money they contributed), how many nationalities attended, how bookings were distributed between the earliest and the latest (this is really vital for people doing the financial planning) and a whole lot of other things which any intelligent person could suggest. If someone at SigAPL would do this we will be glad to print it in Vector. (Or to read it in Quote Quad.)

If we want to buck the trend, we need to know what the trend has been.

Quick Reference Diary 1995-96

Date	Venue	Event
July 28th – August 2nd 1996	The George Fox Complex, Lancaster	APL96: *Designing the Future*

British APL Association meetings are normally held in the IEE, Savoy Place. Nearest tube outlets: Temple or Embankment.

BCSNet: Free Registration Offer

Open to Specialist Group members who join the BCS as new Affiliate Members

If you join the BCS as an Affiliate Member before the end of October 1995, the registration fees for BCSNet Services will be waived.

BCSNet Gold registration fee is £25+VAT.

BCSNet Lite registration is £10+VAT.

For more information contact BCSNet, 1 Sanford Street, Swindon, Wilts, SN1 1HJ or ring 01793-417426 or FAX 01793-480270 or email netadmin@bcs.org.uk

---

Dates for Future Issues of VECTOR

	Vol.12 No.3	Vol.12 No.4	Vol.13 No.1
Copy date	1st Dec 95	1st March 96	24th May 96
Ad booking	8th Dec 95	10th March 96	31st May 96
Ad Copy	15th Dec 95	20th March 96	7th June 96
Distribution	January 96	April 96	July (at APL96)

APL96
LANCASTER

July 28th – August 2nd 1996

Key Dates for Contributors

This Vector should be mailed with a *Call for Papers* enclosed, but just in case we fail to make it on time, here are the dates which contributors will need to meet:

- Abstracts as soon as possible, and definitely by the **end of November.** Nothing formal is needed, just a quick Email to me (Compuserve 100331,644) or Phil (benkard@aol.com) to let us know that something is on the way.
- **31st December.** Draft papers received by Programme Committee; we intend to have the last draft paper out into the review process by 15th January, with pressure on the reviewers to get everything back by 31st January.
- **31st December.** Workshop and panel proposals (again an informal Email is all we need) to one of the Programme chairs.
- **9th February.** Programme Committee meets; selected abstracts ready for the printer by 15th Feb to begin work on the Invitation
- **21st March.** Final copy of all papers to the Proceedings editor, who will aim to have this away to the printers by the end of April.

If you have potential material which directly supports the major themes of the workshops, please let us know; there is no need for this to go through the traditional 'double-blind' reviewing process (unless you want it to), but we would like to have something sufficiently polished to include in the Proceedings.

For those who have lost the leaflet that was enclosed with Vector 12.1, the major themes will be: Effective use of Nested Arrays; Designing for Windows 95; Communication among Co-operating Systems; Algorithms and other Tools of the Trade. As always, good, relevant papers are always welcome on any topic under the sun — we are budgeting on 24 slots, but this is not an upper limit! More is better, and more variety is better still.

Adrian Smith
APL96 Programme Co-Chair

CORRESPONDENCE

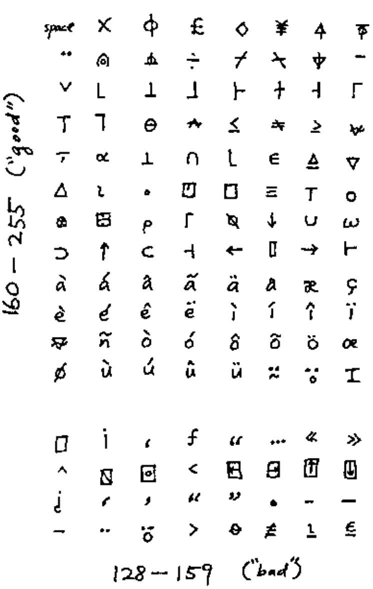
My Computers Are APL Machines

From: Bill Chang Recd at APL95

Proposed Common APL Coding

I can view, enter, edit, print APL from just about any application, including the command line; the first version of this note was posted to comp.lang.apl in March 1994.

Here's my (revised) common APL coding. I've implemented it on the Mac and Sun (xterm). I've found that WWW's HTML file transfer protocol passes 8-bit characters (as opposed to &#NNN) just fine, in accordance with documentation. You should see a table of ISO Latin1 (8859-1) characters. Since I already have APL fonts installed on my machines, I see APL characters, both through news (rn) and the web (mosaic and also lynx, a portable character-based web browser that works great). It works for e-mail too, using the MIME protocol.



Rationale

In the October '94 issue of Vector (11.2 p. 105) Adrian Smith proposed a common APL encoding for Microsoft Windows that includes (most) lower-case national characters but not upper-case. My major critique of this encoding (compatible with APL*PLUS) is that it uses the range 128–159 indispensably. While this is fine in Microsoft Windows or the Macintosh (which can even use most of 0–31), many network gateways and modem / terminal programs either treat 128–159 as 0–31 control characters or mis-handle them in other ways. For example, xterm (X windows) refuses to display 0–31 and 128–159. The well-established ISO Latin1 (8859-1) encoding avoids 128–159. Almost all modern computers already support this 8-bit standard, and mail/news/web have come very close as well. As a result, we can expect 8-bit characters (at least 160–255) to be freely usable and transmittable. I believe APL can hitch a ride with very little effort. I have created a new encoding that places "essential" APL glyphs outside the "dead characters" 128–159, as much as possible.

Software Availability

I can provide the following on request:

- PC/Windows fonts. (DDE keyboard that works in all applications is in progress.)
- Macintosh screen font and custom keyboard mapping (drag-and-drop into system) that work in all applications.
- The popular shareware terminal program Zterm 0.9, patched to use APL font.
- Instructions for making APL the system font (instead of Monaco), so the Mac becomes an "APL machine".
- Sun (X windows) screen font (BDF format) and keyboard mapping (xterm resources) that can be used in all xterm-based programs, such as the command shell, unix utilities, and text editors.
- Simple installation instructions.
- A re-encoding of Adrian Smith's APL2741 (Type 3) PostScript font.
- Printing instructions and utilities.

(Some are not yet finished but are trivial.)

The custom keyboard is *bona fide* APL — Caps Lock toggles between "unified" and "standard" APL keyboards. The font originated in 1987 as a hand-edited, highly optimised version of Courier/APL*PLUS that has the same bitmap size as the Mac's system font (Monaco 9). Over the years it has been revised and re-encoded for APL2 (RS/6000), VAXAPL, and APL.68000. I'm also working on finishing a larger bitmap font and possibly a TrueType font, as time allows.

Please comment. Can someone port this to the PC?

Bill Chang (wchang@acm.org)
Tel: +1 (516) 367-8866

Some Comments

by Adrian Smith

I very much want to be able to cut and paste directly between my APL session (or editor), my word processor, and my Web browser. I can use Bill's suggested layout in Dyalog/W with no problems, but in APL*PLUS III I am stuffed, as there is no way of getting at their mapping table. Does this matter? Readers please advise!

A couple of small quibbles — Bill has put ⍷ in the 'bad' area along with ⍬ and ≠. I can live without ⌶ and ⍫ (would you ever send someone a locked function over the net?!) and possibly even ⍞ (sorry, SigAPL) but I make heavy use of ⍬ and increasing use of ⍷. Some kind of straw poll looks a good idea before we nail this one down for good by offering the fonts and .DOTs on an APL Web page.

With these provisos, I would be delighted to offer the APL2741 TrueType design to the agreed encoding, and also to provide a bitmap session font (and appropriate web.dot) for Dyalog APL.

British APL Association News

from Sylvia Camacho

Members may be interested to know what the BAA is doing about APL 96. As it is to be held in Lancaster one would expect the local APL organisation to be involved. Since the very first international APL conference these conferences have always been controlled by and usually financed by SigAPL, which is a special interest group of the Association for Computing Machinery (usually known as the ACM), the United States equivalent of the British Computer Society.

Below is the text of the reply from our chairman, sent on 22 September to Bob Brown the SigAPL conference co-ordinator.

> *Thank you for your email requesting commitment from the British APL Association to APL96 at Lancaster.*
>
> *Several members of the BAA returned from APL95 with the understanding that a budget for the proposed APL96 would be received in early July. The BAA committee met on 7th July, but no budget had materialised and so no decision could be made, as (like ACM) the BCS does not commit funds without a formal budget and proposal.*
>
> *We did, however, give our backing to the programme proposed by Adrian Smith and resolved to give financial backing to a conference based on that programme as soon as SIGAPL had produced the required budget.*
>
> *As you may be aware, a year ago the BAA declined an invitation to organise APL96. Those of our members who have the necessary skills are heavily committed elsewhere at this time and so we cannot take the lead in this project.*
>
> *Financially, we need a budget that breaks even with a relatively small number of delegates, as we believe that a conference in the UK with such a short lead time is unlikely to attract large numbers. If a suitable budget is produced, and Adrian is still willing to organise the programme, we will make BAA funds available as soon as SIGAPL does the same.*
>
> Alan Mayer
> Chairman, British APL Association.

News from Sustaining Members

Compiled by Gill Smith

Insight Systems

Insight Systems is proud to present two new Client/Server products, the *SQAPL ODBC Server* and *APL Pipes*, which make it possible to integrate application servers written in APL with other development tools, using industry standard interfaces and communications products:

- The *SQAPL ODBC Server* allows your APL application or database system to be accessed as an ODBC data source. You can present your APL-based data or the results of analyses as a set of relational tables which can be used as input into other reporting packages. It is also possible to use application generators such as Borland Delphi or Microsoft Access to create front-ends to APL applications via ODBC. The APL application must run under Unix or Windows NT, but becomes accessible from almost any end-user application or development tool on virtually any platform.

- With *APL Pipes*, you can write distributed APL applications without the overhead of a relational interface such as ODBC or SequeLink. APL Pipes is a TCP/IP based system which is similar to Shared Variables, except that it supports sharing of data and function calls between APL2, APL*PLUS II or III, Dyalog APL and SHARP APL systems running under Windows, Windows NT, OS/2 and Unix. Before the end of the year we plan to add Microsoft Visual Basic and Borland Delphi to the list of *APL Pipes* clients. This will mean that developers in these environments will be able to effectively share variables and make function calls to almost any APL application, whether this application is running on the same machine or accessible over the network.

Until the end of the year, Insight Systems will be very busy with the completion of the Windows version of the KPS System for Adaytum KPS Software Ltd. KPS is a multi-dimensional planning system written in APL, and is currently the UK market leader for budgeting software. We are looking forward to developing the Client/Server versions of KPS scheduled to follow the Windows product, which will allow us to practise our Client/Server preaching and enable APL-based solutions to compete head on with mainstream products in a highly competitive market.

Dyadic Systems Limited
Announcing Dyalog APL/W Version 8

What is Version 8?

Dyalog APL/W Version 8 is a completely new implementation of Dyalog APL/W that is designed specifically for Windows 95 and for Windows NT Version 3.51. It is fully compatible with the current release of Version 7 (7.1 Release 2), but contains many enhancements to support the Windows 95 user interface and other features.

Version 8 is based on Microsoft WIN32 in place of the equivalent Watcom software used for Version 7. WIN32 provides improved memory management under NT and is a key requirement for the new Windows 95 logo for which the product is intended to qualify.

Version 8 includes support for most of the new GUI objects and features provided by Windows 95 and Windows NT 3.51. These include *ListView, ProgressBar, PropertySheet, RichEdit, Spinner, TrackBar* and *TreeView* objects. All new and existing GUI controls have also been enhanced to support the drag-drop feature of Windows 95. This permits the user to drag-drop file icons into a Dyalog APL/W application. It is intended that the new version of Dyalog APL/W will also support OLE2 and .OCX custom controls.

Availability

Dyadic expects to be in a position to ship Version 8 by the end of 1995. However, an early release will be available by the end of September that includes much (if not all) of the additional functionality. Customers may obtain this release by enrolling in the Version 8 Preview Program (see below)

What is the policy for 3.1 and 95?

For the forseeable future, it is likely that many customers will continue to use Windows 3.1. Others will migrate immediately to Windows 95. To cater for both requirements, Dyadic intends to maintain two separate versions of Dyalog APL/W for as long as the requirement to do so exists.

Version 7 will continue to be maintained for Windows 3.1 and OS/2 users. Version 7 is also supported under Windows 95, but it is NOT supported under NT. Customers who need to run applications under Windows 3.1 or OS/2, should remain with Version 7. Applications based upon Version 7 will also run under Windows 95, but will not be able to take advantages of the new Windows 95 features.

Version 8 is Dyadic's product for Windows 95 and NT 3.51 and includes support for many of the special facilities provided by these systems. It is not supported under Windows 3.1. Version 8 is intended to be the state-of-the-art APL for Windows and will be the basis for ongoing developments in the future.

Prices

Version 7 and Version 8 will be marketed and supported as separate products. Version 8 is not provided as an upgrade from Version 7. However, there will be a reduced price for customers purchasing both Versions and for customers who already have Version 7 and purchase Version 8 in addition. Please contact Dyadic or your local Dyalog APL distributor for details.

Version 8 Preview Program

The Version 8 preview program is intended to allow customers to begin to develop Windows 95 (and NT 3.51) applications immediately, without having to wait for the final release of the Version 8 product. It also provides a basis for customers to influence development. This program is open only to existing Version 7.1 users and to any customers who purchase Version 7.1 prior to the final Version 8 release until further notice. The Version 8 Preview Program is chargeable but the price includes a copy of the final release which will be supplied as soon as it becomes available. Enrolling in the Preview Program is the cheapest way for existing Version 7.1 users to obtain Version 8. Contact Dyadic or your local Dyalog APL distributor for details.

During the Preview Program, Dyadic will supply a stream of test releases as new functionality is added and as bugs in the existing code are fixed. In addition to bug reports, subscribers may submit suggestions and requests for enhancement to the new Windows 95 and NT features as they are developed. Dyadic does not guarantee to incorporate all of the requests in the final version, but will endeavour to meet popular demands.

Summary of New GUI Features

Version 8 provides 10 new objects, over 30 new properties and over 40 new events. These new features are summarised below.

- The *PropertySheet* object is a dialog box that allows the user to view and edit the properties of an item. A Standard PropertySheet contains one or more overlapping child windows represented by *PropertyPage* objects, each containing controls for setting a group of related properties. Each PropertyPage has a tab that the user can select to bring the page to the foreground of the PropertySheet. The Display Properties dialog (obtained by clicking the right mouse button on the desktop and choosing Properties) is an example of a standard PropertySheet object. A *Wizard PropertySheet* consists of a sequence of dialog boxes that guide the user through the steps of an operation. In a Wizard PropertySheet, the PropertyPages do not have tabs, and only one page is visible at a time. Also, instead of having **Ok** and **Apply Now** buttons, a Wizard PropertySheet has a **Back** button, a **Next** or **Finish** button, and a **Cancel** button. Windows 95 Setup is an example of an application that uses a Wizard PropertySheet.

- The *ListView* object displays items as a set of icons, in a list format or a report format. 'My Computer' is an example of an application based upon this object.
- The *TreeView* control is a window that displays a hierarchical list of items, such as the headings in a document, the entries in an index, or the files and directories on a disk. Each item consists of a label and an optional bitmapped image, and each item can have a list of subitems associated with it. By clicking an item, the user can expand and collapse the associated list of subitems. 'Windows 95 Explorer' is an example of an application based upon this control.
- The *RichEdit* control is a multi-line text editor that provides extensive word-processing facilities including variable text and paragraph alignment, fonts and colours. This control allows Dyalog APL/W applications to exchange complete documents with Microsoft Word and other word-processors. Data may be imported and exported using .RTF files or via the clipboard. The object also includes direct printer support.
- The *ProgressBar* is a standard output control that indicates the progress of a time-consuming operation.
- A *Trackbar* control is a window that contains a slider and optional tick marks. When the user moves the slider, using either the mouse or the direction keys, the trackbar generates events to indicate the change.
- The *UpDown* control is a pair of arrow buttons that the user can click to increment or decrement a value, such as a scroll position or a number displayed in a companion control.
- The *Spinner* object is a special Dyalog APL object that combines an *UpDown* with an *Edit*. The Spinner permits the user to select from a range of numerical values or text strings.
- The *Grid* object has been extended to allow *Spinner* and *Trackbar* objects to be used to edit cell values.
- The *List* object has been extended. It has a new *MultiColumn* property that allows you to display items in several columns. In addition, the List supports the facility for the user to drag-drop items within it.
- Most controls support the new *AcceptFiles* property. If this is enabled, an object will generate a *DropFiles* event when the user drag-drops one or more file icons onto it from another application, such as 'My Computer' and 'Windows Explorer'.

Causeway Graphical Systems Ltd

Our main item of news is a very simple one: around 3 weeks ago, SAP started cutting our software onto CD to issue with their R/3 3.0a release. From SAP's point of view, 3.0a is a strategic release and is, to the best of our knowledge, the first truly distributed business management system to hit the world market. Perhaps the most encouraging thing about this is that a major player in the world software market has been willing to rely on APL for a critical configuration tool. This has to be a good thing for the future of our language.

This leads naturally to the second item: *Causeway – the Next Generation* (working title: *CTNG*). For the past two weeks, we have had time to relax and work on the future of Causeway rather than hitting the impossible deadlines set by our clients. Since we first released the Causeway utilities in November 1993 (at the British APL Association's BAA-GUI workshop) we have seen a massive increase in the functionality of the Dyalog interpreter. In particular it can now handle resize and reposition (via the *ATTACH* property) in the way that we need it, and it has *Namespaces*. Anyone who has looked through the `Gui_xx` functions will have noticed that a significant slab of code in `Gui_make` is no longer necessary, and may have wondered what all the complex use of ⎕*OR* and ⎕*SHADOW* is doing in `Gui_exec`.

Rather than attempting a piecemeal upgrade, we decided to take the bull by the horns and do a top-down rewrite. It was important to preserve the dialogue-box structure, and it is clearly essential that CTNG behaves in exactly the same way as Causeway, to the calling application. However the internal structure of the class table will be very different (and will support managed replication of functions between classes), and the implementation of localisation and event-handling has been built entirely around the namespace model.

The main visible difference is a huge reduction in the lines of APL code needed to support form-level localisation of variables, and local functions on objects. The entire set of `Gui_xx` functions written so far can be listed on 3 sides of A4, and most of these will never appear outside `#.Gui`. In particular, all the Causeway 'modules' (such as the *Rain* PostScript interpreter) simply become namespaces, and the code in them can be executed much more simply with statements such as `ps.FullScreen` replacing `ps Gui_do'FullScreen'` in application code.

We are on course to have the first release of CTNG included with Dyalog 8; however we also intend to support it under Dyalog 7.1.2 for users who prefer to stay with Windows 3.11 for the moment.

THE EDUCATION VECTOR

October 1995

Editor Ian Clark

This Education Vector has been reprinted from VECTOR Vol.12 No.2. VECTOR is the Quarterly Journal of the British APL Association. For more information about the British APL Association, please contact: Sylvia Camacho, 11 Auburn Road, Redland, BRISTOL, BS6 6LS. Tel: 0117-973 0036.

Contents

Ian Clark
IAC/Human Interfaces,
9, Hill End, Frosterley,
Bishop Auckland,
Co. Durham DL13 2SX.

Tel: 01388-527190
Email: 100021.3073@compuserve.com

Editorial
Raising Standards — or Flying Kites?

If you are one of those people who believe that reading newspapers, like tarot cards, confers insight into reality, you cannot fail to have noticed that something's the matter with our universities this year. They're being swamped by a tide of mediocrity.

The trouble started with the A-level examination results. Good grades were achieved by too high a proportion of school-leavers, which entitled an excessive number to apply for university admission. Since the human genome doesn't change that much in the space of a few years, this could only come about, according to some, by a general relaxation of standards.

There is another explanation, which the government has been surprisingly slow to latch onto. It is that the painful re-engineering of public education and welfare over the last few years has at last borne fruit in a cohort of better educated students, additionally motivated by a shot of genuine fear for their future prospects. Students, indeed, who are more able to jump the traditional hurdles designed to filter the candidates for tertiary education.

Perhaps the government doesn't believe this (a cynical standpoint, in view of their efforts) but suspects the hurdles have been lowered out of self-preservation by unscrupulous academics. But whom exactly do the hurdles filter out, and why?

The traditional concept of matriculation and its successor, the A-level examination, is founded on the premise that only a proportion (from a tenth to a half) of people who put themselves forward for higher education ought to be accepted for it. Otherwise "standards fall".

There are two views about why standards should fall. The first is that resources are too limited to allow everybody who wants tertiary education to get it, which invites calls for an expansion of places on degree courses. The second is that you don't want to over-educate the duffers. This invites people to believe that there is some rational criterion being applied to the acceptance and rejection of candidates, a belief which doesn't bear close examination.

Of course it is open to the more sought-after universities (whatever they imagine they are doing) to select those candidates who can tolerate old-fashioned teaching of patchy quality — in effect teach themselves. This frees the tutors to concentrate on research, consultancy, authorship, cutting a dash in public and other more rewarding activities than the prime purpose of paying them out of the public purse — to train the nation's youth. Other establishments ape their betters, hoping to avoid the slur that their only role in life is to scavenge the rejects.

But do rational rejection criteria really exist, let alone are they being applied? A longitudinal study of student performance at my old place uncovered no significant correlation (in fact zero) between students' grades on admission and their eventual class of degree. Granted there are many caveats surrounding this kind of study, it did raise an alarming question. When we selected candidates on the basis of their grades, just what did we imagine we were doing?

Are there no predictors, then, of a candidate's eventual performance? No, the study didn't conclude that. Indeed certain syndromes were associated with what might be called (again with a legion of caveats): "losers". These are the hard-do'ers and the satisficers, who are capable of putting their tutors, counsellors, colleagues — indeed the whole institution — to enormous trouble on their behalf, at small cost to themselves. The sort of person you don't get to help you carry a ladder, or you end up carrying it yourself, plus him or her dragging along on the other end. Purely on cost-saving grounds, these students amply repay weeding out at the earliest stage.

But that accounts for no more than 5% of the intake. For the rest, it's not so much a creaming-off process, more a thinning out, like seedling in a tray. Gardeners among us will know the rational basis for this. It is done like decimating a Roman legion, because its purpose is not to select the "best" (most of the seedlings look alike) but to relieve overcrowding.

What is the evidence for overcrowding in today's educational market? It all depends on the course. Romantic ones like Veterinary Science, or lucrative ones like Law, are heavily oversubscribed. For Mathematics, Physics, Chemistry and Engineering, all the signs point to massive overcapacity. In the Tyne-Tees area alone there are now five universities, whereas purely to serve the region's needs for people educated to degree level you'd be hard-put to make a case for one. Higher education has simply replaced coal and ships as a commodity for export, not to mention occupying the unemployed masses.

Yet we read [Independent, Thursday 31st August, p4] that "7 thousand students are admitted to University each year through foundation courses which do not always require them to have A-levels. Mrs Gillian Shepherd said applicants must be told they had to meet rigorous standards to ... embark on a 3-year degree".

This is revealing of an attitude to University entrance devoid of rational basis. To the youth of today, Mathematics, Physics, Chemistry and Engineering sound like a lot of hard work for pretty small rewards. Is it any wonder that courses are under-subscribed and departments face closure? It's a matter of public concern (or scandal, depending on how you look at it) that a high proportion of engineering degree students need remedial Mathematics at the how-to-balance-brackets level. But it's only being realistic to admit such ill-trained candidates, in circumstances that have arisen over the years through public neglect or even deliberate policy. When there's been a frost in the greenhouse, you don't thin out the seed-tray. You coddle whatever germinates.

Some Articles in the Current Vector

J-ottings 7

by Norman Thomson

With the release of the J Release 2 interpreter as Freeware (hereafter referred to as JFW which conveniently stands for either J FreeWare or J For Windows), J has reached a crossroads in more ways than one. In the first place, the amateur / education / non-commercial J user who it is assumed might be reading this column, must decide whether to opt for J7 (that is the last release of DOS J) or for JFW. If he/she cannot, or does not, run Microsoft Windows, then the choice is of course made. Where the choice is available, my vote is for JFW, primarily because it includes control structures. However, DOS J fanciers should not stop reading; a template for the equivalent form of a simple loop in a dyadic defined verb is the following:

JFW

```
fn=.3 :0
''
:
 initialisation
while.  condition do.  body
end. result
)
```

J7

```
fn=.0 :0
''
:
 initialisation
lab) $.=.>( condition ){end;$.
 body    [ $.=.lab
end) result
 )
```

As an example, here are two equivalent defined verbs which compute continued fractions iteratively, e.g. 2 + 1/(2 + 1/(2 + ... 100 times. The left argument is the number of iterations, the right argument is the start value for the fraction, so that for this example `x. = 100` and `y. = 2`.

JFW

```
fn=.3 :0
''
:
r=.y. [ i=.0
while. i<x.
  do. r=.(+%)r
      i=.i+1
end. r
)
```

J7

```
 fn=.0 :0
 ''
 :
 r=.y. [ i=._1
lab) $.=.>(x.>i=.i+1){end;$.
 r=.(+%)r
 $.=.lab
end)r
 )
```

The JFW version shows what might be regarded as a "nice" layout for the loop — for those who like maximum compression, the loop body could be written

```
while.i<x.do.r=.(+%)r[i=.i+1 end.
```

— shades of APL one-liners, aided by the use of the verb [as statement separator. It is stressed that the above verb is written to demonstrate a loop — if the result alone were the goal, then the natural J way to write it would be to use the power adverb

```
(+%)^:100(2)
```

To test whether the benefit of control structures is paid for by performance overhead, an "empty loop" (monadic) was implemented under both interpreters.

JFW

```
fn=.3 :0
i=.1
while. i<y.
$.=.>(y.>i=.i+1){end;$.
  do. i=.i+1
end.
)
```

J7

```
fn=.0 :0
i=.0
lab)

$.=.lab
end)i
:
''
)
```

(Note: In J7, it is necessary to define the null dyadic option explicitly, but this is not the case in JFW.)

The JFW version runs almost twice as fast as the J7 version, indicating that the control structures per se give a performance improvement. However, with some algorithms, J7 is faster than JFW — all that can be said with certainty is that the two interpreters are different.

Apart from control structures, the other major difference between J7 and JFW is the definition of *amend*. This is illustrated using one of the simplest numerical algorithms which cannot be expressed without a loop — even in J! Choleski computes the "square root" of a square symmetric matrix, that is, for a given matrix **M**, the Choleski matrix **C** has the property that **C'C = M** where dash denotes transpose.

J7

```
   tr=. |:                              NB. transpose
   ip=. +/ .*                           NB. inner product
   sq=. ip~tr                           NB. "square" of matrix, i.e. M'M
   csq=.[ { ({sq)                       NB. inner prod of one column of matrix
   rd=.(>:@[) }.{                       NB. drop i+1 elements from ith row -
                                        NB.  rd stands for "right of diagonal"

   choleski=.0 : 0
i=._1 [ z=.($y.)$0                           NB. Initialization
lab) $.=. >((#y.)=i=.i+1)($.;end             NB. Start of loop
z=.(%:( (t=.<i,i){y.)-i csq z)(i *>:#y.  )}z NB. Diagonal element
z=.(((i rd y.)-i rd sq z)%t{z)(i rd i.$y.)}z NB. Assign to rt of diag
$.=.lab                                      NB. Back to start of loop
end)z                                        NB. Publish result
:
''
)
NB. (i rd sq M) is products of col i of M with all cols to its right

   tm=.3 3$10 2 4 2 5 3 4 3 9
   ]um=.choleski tm
3.16228 0.632456 1.26491
      0  2.14476 1.02576
      0        0 2.51949
   tm-:(tr um)ip um
1
```

JFW

Instead of rd define

```
   each=.&.>
   rdi=.(>:@[ }.,each)i.@#         NB. rdi = "right of diag" scatter inds
   choleski=.3 : 0
i=._1 [ z=.($y.)$0                              NB. Initialization
while. (#y.)>i=.i+1 do.                         NB. Loop through rows
 z=.(%:(t{y.)-i csq z) (t=.<i,i) }z             NB. Amend diagonal element
  if. i~:_1+#y. do.                             NB. Except for last one,
amend
   z=.(((p{y.)-p{sq z)%t{z) (p=.i rdi z) }z NB. elements right of diag
  end.
end.
)

NB. (i.0)(i.0)}z is valid in J7 but NOT in JFW, which provides an
opportunity for introducing the if. control word in the above.
```

Compare the third and fourth lines in J7 with the third and fifth lines in JFW, all of which update z using amend. The overall structure of amend (see *J-ottings 6*) is the same in both JFW and J7, namely

```
(data)  (indices)  }  (data object)
```

However in J7 "indices" are interpreted as "linear indices", whereas in JFW "indices" means a sequence of boxed scatter indices, thus making indices consistent across both { and } .

The verbs `rd` and `rdi` share the same fork structure:

```
( (>:@[)  }.  v )     (where v = { in J7,  ,each in JFW)
```

The composition `>:@[` means "add one to the left argument", which in composition with `}.` (drop) identifies elements to the right of a matrix diagonal.

In the fourth line of the J7 version, `rd` is used repeatedly with different matrices. In particular, within the indices component of amend, it is used with `i.` to produce linear indices. In JFW `rdi` produces the corresponding scatter indices. Both `t` and `p` are assigned within indices components, and then used for selection in the corresponding data components, thereby demonstrating index consistency.

The arrival of JFW raises broader issues concerning the educational value of J as an executable notation. In the days when APL was trying to make its way in the computing world, the smart way to condemn it without going to the trouble of finding out anything about it was to repeat the glib assertions:

"Needs special equipment";
"Costs a lot"; and
"Hasn't any control structures".

JFW has removed all of these objections, and it will be interesting to observe the excuses which the computer science fraternity will doubtless bring forward for ignoring J in its turn.

The potential of APL as an executable notation for conveying mathematical and more general data-structuring ideas has long been proclaimed by small bands of enthusiasts around the world. Now that the above objections have been removed, JFW can provide even greater generality and power for achieving the above goals. Howard Peelle at the University of Massachusetts, has for the purpose of in-service teacher training produced admirable workshop material for *Teaching Mathematics with J*. Briefly the principle is to introduce a minimal amount of J, and use the combination of experimentation, discussion and reflection to focus attention on mathematical concepts. However, efforts of this sort in curriculum development need encouragement and funding to sustain enthusiasm.

In many ways the ground is less fertile now than it was in the APL days. First, computer algebra systems such as *Maple* and *Derive* have achieved considerably greater popularity than executable notations. Unquestionably these have brought many conceptually simple but computationally heavy problems within reach of students, which is fine if results as opposed to methods are the primary goal. Secondly, the pendulum of educational fashion seems to be swinging away from knowledge towards concepts such as "Vocational Qualifications" and "Generic Skills". (Arguably data-structuring and data-manipulation are themselves generic skills — given the laxity of spelling these days it might even be possible to promote them as J-eneric skills!)

Thirdly, in the UK, teaching by administrative overload seems to be taking over at all levels from teaching by inspiration. Consistent uniformity, and customer delight elicited by satisfaction surveys are of course to be applauded when mass-producing manufactured items, and such qualities are rightly recognised as indicators of excellence in this domain. However, it is disturbing to see the same indicators being accepted in an uncritical way in the education process, where excellence ought to mean something of a quite different sort. Cambridge University, for example, has long enjoyed a reputation for providing an environment in which budding excellence in mathematics can flower. However, in the words of the Professor of Applied Mathematics and Theoretical Physics there, there has been since 1990 a steady but quite rapid deterioration in the levels of preparedness of first year students which means that it is no longer possible to teach all the material. We must, I suppose, be grateful that the Greeks never hit on Total Quality Management, which if in place, would presumably have put paid at an early stage to the careers of Archimedes, Pythagoras and their like!

Such matters heighten the significance of the crossroads at which J has now arrived. Who and what is J for, and what should its place be in the broader spectrum of science? — these are questions which are now ripe for debate. Please send your comments to the editor.

Two Numerical Algorithms in J

by Antje Muller, Tineke van Woudenberg and Alistair Young

Polynomial Interpolation

Interpolation means the process of using a table of values of an independent variable *x* with corresponding function values *f(x)*, to determine a value of *f(x)* at a non-tabular value of *x*. For many functions, a polynomial which passes through the tabulated points gives an adequate representation near the non-tabular value. Such a polynomial is then used to interpolate values of *f(x)* at other values of *x* without explicit calculation of its coefficient values.

There is a variety of techniques for doing this, among them Lagrangian polynomials and Neville's algorithm. The technique used here is known as **Newton's Divided Differences**, and consists of taking successive differences of f-values which are divided at each stage by the matching differences in the column of x-values. The values of the nth column of f-differences are divided by the differences of the corresponding x-values which are n apart, for example:

```
x       f       1st      2nd       3rd
                diff     diff      diff
0       4
                 2
1       6               0.2
                 3                -0.15
5       18               -1
                -4
8       6
```

In the 1st diff column 18-6 is divided by 5-1, and 6-18 by 8-5. In the 2nd diff column, 3-2 is divided by 5-0 and -4-3 by 8-1, and so on.

The result of the verb `divdifs` is x laminated with the values at the head of the columns, viz.

```
0 1 5    8
4 2 0.2 _0.15
```

from which the original values of `f` could, if required, be reconstructed by reversing the arithmetic.

The J realisation which follows is in the style of a Pascal program, which is a natural way of expressing an iterative algorithm of this sort. Indeed this algorithm illustrates the ease with which transition is possible between J and more familiar programming languages.

```
NB. Obtain leading values in divided difference columns
divdifs =. 0 : 0
:
i =.0 [ f=. a=. y. [ n=.#x.                NB. initialization
loop) $.=.>(n > i=.i+1) { end;$.           NB. for i = 1 to n-1 do
j =. i-1                                   NB. start of inner loop
loop1) $. =. >(n > j=.j+1) { end1;$.       NB. for j = i to n-1 do
t =. (j{f - (j-1){f)%(j{x. - (j-i){x.)     NB. a[j]:=
a =. t j } a                          NB. ((f[j]-f[j-1])/(x[j]-x[j-i])
$.=.loop1                                  NB. end of inner loop
end1) f =. a                               NB. update f
$.=.loop                                   NB. end of outer loop
end) x.,:a                            NB. result is x,:leading diff'nces.
)
```

Interpolation is performed by an algorithm which uses the leading divided differences. A merit of this technique is that as the algorithm progresses through its iterations the result vector grows by catenating interpolated values of successively higher polynomial degree as more points are brought into consideration from the right of the data.

```
NB. Program to perform polynomial interpolations of increasing order
interpol =. 0 : 0
''
:
f =. 1{x. [ x =. 0{x.           NB. f = function values, x = x-values
a =. 1{t =. x divdifs f         NB. a = leading divided differences
n =. #a [ i =. 0 [ p =. 0{a     NB. p =a[0], first estimate
sol =. a                        NB. sol is result vec (see last NB.)
loop) $. =. >(n>i =.i+1) { end;$.     NB. for i = 1 to n-1 do
j =. i [ t =. 1                       NB. start of inner loop
loop1) $. =. >(_1<j=.j-1) { end1;$.   NB. for j = i-1 downto 0 do
t =. t*(y. -j{x)                      NB. t := t*(y. -x[j])
$. =. loop1                           NB. end of inner loop
end1) p =. p+t*((i){a)                NB. p:=p + t*a[i]
sol =. p i } sol                      NB. sol[i]:=p
$.=.loop                              NB. end of outer loop
end) sol                    NB. sol is successive polyn'l interpolns.
)
```

```
      x =.8 9 9.5 11
      f =. 2.079442 2.197225 2.251292 2.397895
      xf =. x,:f
      xf interpol 9.2
2.079442 2.22078 2.21924 2.21921
```

The last line shows that the cubic interpolation is 2.21921, although this is little different from the value obtained by quadratic interpolation.

It is not necessary that the x values be in ascending order. For example, using the first example given above the successive polynomial interpolations are

```
4 10 11.2 13
```

If the points are introduced in x-value order `0 8 1 5`, the polynomial interpolations are

```
4 4.75 8.5 13
```

which reflects the fact that the straight line approximation will vary greatly according to when the point (5,18) is introduced.

Choleski Decomposition ("Square Root") of a Matrix

This is a method of factorisation of a symmetric matrix

$$\mathbf{A} = \mathbf{L}\mathbf{L}'$$

where **L** is a lower triangular matrix. If **A** is positive definite the elements of **L** will be real, otherwise some will be complex. The mathematical equations which govern the method are:

$$m_{11} = \sqrt{a_{11}}$$

$$m_{jj} = \sqrt{a_{jj} - \sum_{s=1}^{j-1} m_{js}^2} \quad j = 2,\ldots,n$$

$$m_{j1} = \frac{a_{j1}}{m_{11}} \quad j = 2,\ldots,n$$

$$m_{jk} = \frac{1}{m_{kk}}\left(a_{jk} - \sum_{s=1}^{k-1} m_{js} m_{ks}\right) \quad k+1,\ldots,n; k$$

These lead to the definition of some subsidiary verbs as follows:

```
ssqbut1=. +/@( (-:i.@#@]) # *:@] )       NB. 2 ssqbut1 4 2 7 3 5 yields the
                                         NB. sum of squares of 4 2 3 and 5
vb=. %(%:@{.)                            NB. vb 2 3 4 yields
                                         NB. sqrt(2), 3/sqrt(2), 4/sqrt(2)
stap1=. (vb@{.)&.|:                      NB. performs vb on first column
                                         NB. of a matrix
column=. (*i.)@#                         NB. column yields 0 n 2n 3n ...(n-1)n
                                         NB. where n=#y. (y. is a matrix)
fillmatrix =. $~$                        NB. x. fillmatrix y. yields matrix of
                                         NB. same size as y. filled with x.
fillM1 =. '' : ' (stap1 y.) (column y.) } (0 fillmatrix y.)'
                                         NB. 1 fillM1 m yields zero
                                         NB. matrix with first column
                                         NB. containing the result of stap1
```

As with `interpol` a Pascal program is a natural way of expressing the essentially iterative algorithm, whose development, given the above verbs is now straightforward:

```
NB. PROGRAM choleski
ch =. 0 : 0
k=. 0 [ n=.#M=. 1 fillM1 y.              NB. initialisation
loop1) $. =. >(n>k=.k+1){ end1;$.        NB. if k >= n go to end1
t=. %: k{k{y.- (k ssqbut1 (k{M))         NB. t = sqrt(Akk - sum(Mks^2)
M=. t((n+1)*k) } M                       NB. Mkk:= t
j =. k                                   NB. start of inner loop
loop2) $. =. >(n>j=.j+1) {end2;$.        NB. if j>= n go to end2
t=. (j{k{y.- (k{M) +/ .* j{M )% t        NB. t:=(Ajk - sum MjkMks)/Mkk
M=. t ((j * n) + k) } M                  NB. Mjk:=t
$. =. loop2                              NB. end of inner loop
end2)                                    NB. j=n
$. =. loop1                              NB. end of outer loop
end1)M
:
''
)
```

Two examples are:

```
    A                 ]x=.ch A                   x+/ .* |:x
 4 2 14                2  0 0                        4  2 14
 2 17 _5               1  4 0                        2 17 _5
14 _5 83               7 _3 5                       14 _5 83

    a                     ch a
 1 4 6                 1    0            0
 4 0 3                 4    0j4          0
 6 3 2                 6    0j5.25       0j2.53722
```

The authors of this article are post-graduate mathematics students at the Universities of Kaiserslautern, Eindhoven and Strathclyde respectively, and were introduced to J version 7, and to each other, in the course of ECMI (European Consortium for Mathematics in Industry) exchanges at the University of Strathclyde.

Technical Note on Matrix Decomposition

by Norman Thomson

J contains, as a primitive, `128!:0`, the so-called QR decomposition of a square matrix. This note describes how to obtain this in APL, and also how to obtain the LU decomposition where L and U stand for Lower and Upper triangular matrices, in the latter case with 1s on the diagonal. L and U can if required be merged compactly into a matrix of the same shape as the orginal matrix by overlaying all those cells which, by definition, must be either 0 or 1.

QR decomposition is also known as *Gram-Schmidt orthogonalisation*. The comments on the functions should make the procedures clear, particularly if used alongside the sort of algebraic descriptions of the algorithms which can be found in many texts on Numerical Analysis such as *"Matrix Computations"* by Golub and Van Loan, or the *"Numerical Recipes"* series. Perhaps surprisingly, it is in the nature of the algorithms that they are not particularly helped by the availability of the array operations of APL.

```
     ∇ Z←QR R;T;U;I;N
[1]  ⍝ R is square.  R≡⊃+.×/Z, cols of ↑Z are orthonormal, 2⊃Z is upper Δr
[2]    N←↑ρR ◇ Z←2ρ⊂(N,0)ρI←0              ⍝ initialize, Z is built col by col
[3]   L1:→(N<I←I+1)/0 ◇ T←R[;I]            ⍝ begin loop, NORM is sqrt +/x*2
[4]    Z←1E¯6 CLEAN Z,¨(⊂U÷NORM U),⊂N↑T,NORM U←+/(1,-T←T+.X×↑Z)×[2]T,↑Z ◇ →L1
     ∇
```

```
     ∇ Z←NORM R
[1]   Z←(+/R*2)*0.5                        ⍝ Euclidean norm
     ∇
```

```
     ∇ Z←L CLEAN R
[1]   Z←R×L≤|R                             ⍝ sets small numbers to zero
     ∇
```

```
     ∇ Z←LU R;I;N;T;K;U
[1]  ⍝ ↑Z is lower Δr, 2⊃Z is upper Δr with diag 1s.  R≡⊃+.×/LU Z
[2]   Z←R ◇ U←⍳N←↑ρR ◇ I←K←0                              ⍝ initialize for loop[3-6]
[3]  L1:→(N≤I←I+1)/L3 ◇ T←I↓U ◇ →(I=1)/L2                 ⍝ first row is special case
[4]   K←Z[,I;⍳I-1]+.×Z[⍳I-1;T]                            ⍝ K is for use in [5]
[5]  L2:Z[I;T]←(R[,I;T]-K)÷Z[I;I]                         ⍝ set Ith row to rt of diag
[6]   Z[T;I+1]←R[T;I+1]-Z[T;⍳I]+.×Z[⍳I;I+1] ◇ →L1         ⍝ diag+below of (I+1)th col
[7]  L3:Z←((U∘.≥U)×Z)((U∘.=U)+(U∘.<U)×Z)                  ⍝ omit [7] for compact form
     ∇
```

```
      M
 8 2 1 3
 1 5 1 8
10 3 2 4
 9 7 2 7

      QR M
0.5101   ¯0.2921 ¯0.5667  0.5774    15.68 7.268 2.997  8.607
0.06376   0.7761  0.2456  0.5774    0     5.846 0.8929 7.602
0.6376   ¯0.2796  0.7179  0         0     0     0.4723 0.8879
0.5738    0.484  ¯0.3212 ¯0.5774    0     0     0      2.309

      LU M
 8 0    0       0    1 0.25 0.125   0.375
 1 4.75 0       0    0 1    0.1842  1.605
10 0.5  0.6579  0    0 0    1      ¯0.84
 9 4.75 0      ¯4    0 0    0       1
```

APL Product Guide — Full

compiled by Gill Smith

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.

Pressure on space sometimes prevents us from printing the complete guide, however updates will always be listed. We do depend on the alacrity of vendors to keep us informed about their products. Anyone who is not included in the Guide should contact me to get their free entry — see address below. If we have your Email address listed, we will mail you a reminder in good time for the next issue.

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.

For convenience to readers, the product list has been divided into the following groups ('poa' indicates 'price on application'):

- Complete APL Systems (Hardware & Software)
- APL Interpreters
- APL-based Packages
- APL Consultancy
- Other Products
- Overseas Associations
- 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.

We also welcome information on APL clubs and groups throughout the world.

All contributions and updates to the APL Product Guide should be sent to Gill Smith, at Brook House, Gilling East, York, YO6 4JJ. Tel: 01439-788385, Email: 100331.644@Compuserve.com

COMPLETE APL SYSTEMS

COMPANY	PRODUCT	PRICES(£)	DETAILS
Dyadic	IBM RS/6000 MD320	11,736	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)
Interprocess Systems	APL2 Dev't Workstation	poa	Mainframe APL2 supported on a PS/2 via a co-processor card with 16Mb of memory running VM/ESA (370 mode). A complete system includes a PS/2, a P/370 co-processor card, and software licenses for VM/ESA, APL2, GDDM and the full line of Interprocess APL2 enhancements.
Optima	IBM Compatible	poa	Complete PC-based station, APL interpreters & all support eq't

APL INTERPRETERS

COMPANY	PRODUCT	PRICES(£)	DETAILS
APL Software	APL*Plus/PC Release 10	450	STSC's APL for IBM PCs & compatibles.Upgrades from earlier releases also available.
	Run-time	poa	Closed version of APL*Plus/PC which prevents user exposure to APL
	APL*Plus II	1,395	All the features of mainframe APL*Plus for your 386PC!
	Run-time	poa	
	Dyalog APL	1000-10,000	2nd generation APL for Unix systems
	APL2/PC	poa	IBM's APL 2 for the PC.
Atlantis Software	Analytic Platform (*K*)	poa	*K* is an APL-like language
Beautiful Systems	Dyalog APL/W for Windows	poa	US Distributor of Dyalog APL products from Dyadic.
	Dyalog APL for Unix	poa	See Dyadic listing for product details.
The Bloomsbury Software Company	APL*PLUS PC Rel 11	250	STSC's full featured APL for IBMs and compatibles - Version 11 gives free runtime.
	APL*PLUS III Windows	949	The new 32-bit native Windows APL*PLUS. Develop in Windows, and distribute APL applications with no runtime charges. Reasonable migration charges from APL*PLUS/PC and APL*PLUS II.
	APL*PLUS II for DOS	750	Now that APL*PLUS III for Windows is available, the facility for creating Windows applications in PLUS II has been removed, and the price reduced.

	APL*PLUS II for UNIX	poa	STSC's 2nd generation APL for all major Sparc and Risc Unix workstations.
	APL*PLUS VMS	poa	2nd generation APL for DEC VAX computers 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 Screen Manager. Requires 386/486 based PC or PS/2, at least 2Mb RAM, EGA or VGA, DOS 3.3 or later.
	Dyalog APL/W for Windows	995	As above, plus object-based GUI development tools. Requires Windows 3.0 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.
IAC/Human Interfaces			
	I-APL/Mac	13	Macintosh version of I-APL
I-APL Ltd	I-APL/PC or clones	8	ISO conforming interpreter. Supplied only with manual (see 'Other Products' for accompanying books).
	I-APL/BBC Master	8	As above
	I-APL/Archimedes	8	As above
	Strand Software Inc		Strand Software Inc has the sole selling rights to Iverson Software Inc products. I-APL stocks a few of these (mainly APLIWIN and the personal J products and books), but is no longer an agent.
IBM APL Products	TryAPL2	free	APL2 for educational or demonstration use. Write, fax or Email to APL Products; specify disk size desired.
	APL2 PC (US Version)	$630	Product No. 5799-PGG. PRPQ Number RJ0411. Order from 1-800-IBM-CALL
	APL2 PC (European Version)	£348	Product No. 5604-260. Part number 38F1753. From all IBM dealers, including MicroAPL.
	APL2 for OS/2 Entry Edition	$185	Part No 89G1556.
	APL2 for OS/2 Advanced Edition	$650	Part No 89G1597. Contains all facilities of the Entry Edition plus: DB2 interface; co-operative processing TCP/IP interface; tools for writing APs; TIME facility
	APL2 for Sun Solaris	$1500	Product No. 5648-065.
	APL2 for AIX 6000	poa	Product No. 5765-012.
	APL2 Version 2	poa	Product No. 5688-228. Full APL2 system for S/370 and S/390
	APL2 Application Envt Vn2	poa	Product No. 5688-229. Runtime environment for APL2 packages
Insight Systems	APL*PLUS/PC	poa	APL systems marketed and supported ...
	Dyalog APL	poa	from: Dyadic, Manugistics, IBM
	APL2	poa	under: Windows, OS2 and Unix
Iverson Software Inc.	J Professional (inc runtime)	$495	
	J Personal Edition	$100	
	J Personal (disks only)	$40	The text of the manuals is available as Windows Help, so the paper copies are no longer a necessity.
	APLIWIN	$30	For 386/PC under Windows 3.1
	APL Reference Manual	$30	Documentation for all the above.
	J System Kit	$24	J 6.2 diskette with manual "J:Introduction and Dictionary"
	J Source Code	$90	Full C source code plus 100-page book
MasterWork Software	Manugistics Products and ISI	poa	New Zealand distributor

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/X	1500-6000	Second-generation APL. Nested arrays, user defined operators, selective specification, etc. Multi-user AIX version with full OSF/Motif support.
	APL.68000 Level I		
	Mac, ST, Amiga	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 4.0	1395	
Optima	APL*PLUS/PC	369	
	APL*PLUS II	950	
	APL*PLUS II PC Developers Kit	poa	
	Dyalog APL	999	
RE Time Tracker Oy	APL*PLUS/PC	poa	Complete APL*PLUS and Statgraphics product range and user support for Finland
	APL*PLUS II/DOS		
	APL*PLUS III/WIN		
	APL*PLUS/UNIX		
Soliton Associates	SHARP APL for MVS	poa	for IBM MVS mainframes
	SHARP APL for Unix	poa	for IBM RS/6000 and Sun SPARC
Strand Software	Canada		
	All APL*PLUS Products	poa	All APL*PLUS products including upgrades and educational.
	Dyadic and ISI products	poa	
	USA		
	Dyadic and ISI products	poa	
Uniware	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/UNX	call	STSC's full feature APL for UNIX based computers
	APL*PLUS II	call	STSC's full feature APL for 386 machines.

APL PACKAGES

COMPANY	PRODUCT	PRICES(£)	DETAILS
Adaptable Systems	FLAIR	poa	Finite loader and interactive rescheduler. Customisable full-function scheduling system. (Available outside Australia by special arrangement only.)
APL-385	APL-385 for APL*PLUS/PC	50	including ...
	FSM-385		Screen development
	DRAW-385		Screen design
	DB-385		Relational W.S.
	GEN-385		Miscellaneous Utilities

The APL Group	Qualedi	$1500-4000	Electronic Data Interchange (EDI) translation software for the PC, with strict compliance checking.
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
Beautiful Systems	ASF_FILE	$399	Dyalog APL/W auxiliary processor for access to APL*PLUS/PC APL component files (*.ASF).
	NAT_FILE	$299	Dyalog APL/W auxiliary processor which emulates the APL*PLUS/PC quad-N native file subsystem for access to the DOS file system.
	DBF_FILE	$299	Dyalog APL/W auxiliary processor for efficient block mode access to dBASE format files. Designed to get large amounts of data in and out of dBASE. Not suited for random access to small amounts of data (it does not handle keys).
The Bloomsbury Software Company (for VSAPL)	Enhancements & Sharefile	poa	Component files, quad-functions & nested arrays for VSAPL under VM/CMS & MVS/TSO
	Compiler	poa	The First APL compiler!
(for APL2)	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.
Causeway	Causeway for Dyalog/W	$50	Manuals and Class-management utilities for the Causeway platform supplied free with Dyalog APL/W
	Causeway for APL*PLUS III	$50	Software, class-management utilities and printed documentation for Causeway under APL*PLUS III
	Rain Graphics Workspace	$250	Full on-line documentation (Windows Help) and handy-reference card for the *Rain* business and statistical graphics workspace supplied with Dyalog/W.
Cinerea AB	ORCHART	250	Organization chart package for IBM APL2/PC. Full & heavily commented source code included - free integration into other applications. NB: ASCII output with line-drawing (semi-graphic) characters for boxes.
CODEWORK	HELM	poa	Decision Support system for top management. Developed in Italy over 7 years. Requires APL mainframe or APL*PLUS/II. Optional modules: EIS, Excel interface, DTP output via LATEX, output on map background.
CYBEX AB	APL Graf/PC	290	Presentation graphics for APL*PLUS/PC (CGI)
	APL Graf II/PC	390	Presentation graphics for APL*PLUS II/PC (CGI).
	Utility Functions APL2	1900	For APL mainframe; incl. a very fast search.
	Utility Functions II/PC	130	Same package for APL*PLUS II/PC.
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	Software to transfer workspaces between APL*PLUS and Sharp, and between APL*PLUS and I-APL. Software to import IBM .ATF files to APL*PLUS.
	APL*PLUS Utilities		Public domain software, unlock locked fns, a user-friendly alternative to locking, fns of mathematical physics, menus, and others.

IAC/Human Interfaces	IAC/Graf	15	Graph plotting for I-APL/Mac
	IAC/Vox	15	Spoken APL characters for I-APL/Mac
I-APL Ltd	Educational workspaces	5	PC format disks with the examples from: Thomson, Espinasse (Kits 1-4), Kromberg, Jizba & FinnAPL. All the examples to save your fingers!
IBM APL Products	A Graphical Statistical System	$250	for DOS, Product Number 5764-009
	(AGSS)	$500	for Workstations (OS/2, Aix, Solaris), Product Number 6764-092
		$2500	for CMS, Product Number 5764-011
Impetus Ltd	*Impetus*	poa	Corporate Modelling and Reporting System.
INFOSTROY	APL*PLUS/Xbase Interface (II/386 Version 2)	$198	Complete package written in C. Comparable with the data, index & memo files of FoxPro, dBASE, & Clipper. Multi-user support. No DBMS license required.
	(PC Version 2)	$98	As above for APL*PLUS/PC.
	(DLL Version 1)	$198	The same in a DLL form! Gives your Windows applications all advantages of DLLs.
Insight Systems	IUTILS/XP	20-95	Cross-platform utility library including simple OS calls (DIR, COPY, DEL, RENAME) and DATE functions. For APL*PLUS II, APL2 and Dyalog APL under Windows, OS/2 and Unix.
	ASI	95	APL Spreadsheet Interface. "Device-independent" spreadsheet driver supporting Excel, 123 and Quattro-Pro for Dyalog APL/W
	WinCom	95	Asynchronous comms package for Dyalog APL/W
	S2D,22D,X2X	poa	Advanced APL syntax analysis and conversion packages from Sharp and APL2 to Dyalog, and between any two APLs
	SQAPL Client	poa	Interface from APL*PLUS II, APL2 and Dyalog (Windows, OS/2 or Unix) to most SQL databases over most networks.
	SQAPL Server	poa	Makes APL*PLUS II, APL2 or Dyalog APL (Unix) available as SequeLink servers. Can be called from SQAPL clients or other applications such as Excel, C++, Smalltalk, Visual Basic.
Interprocess Systems	APL2 Development Workstation	poa	
	IEDIT	$3000-5000	Full screen APL2 editor with immediate APL execution, and full-screen debugger
(mainframe)	AFM	$15300	High performance component and keyed file system (VS APL and APL2)
	Enhanced Format	$2575	A QuadFMT data formatter for VS APL and APL2
	PowerCode	$2000	External functions for APL2
	WSORG	$1500	Full-screen Workspace Organizer for APL2.
JAD Software	JAD SMS	150-500	Software management system for APL*PLUS II based on hierarchical databases; includes full-screen interface and stand-alone functions. Price depends on number of users.
Lingo Allegro	FRESCO Business Graphics	poa	Fast and Easy Business Graphics DLL
	AP126/PC	poa	GDDM Interface for Dyalog APL/W
	AP127/PC	poa	ODBC Interface for Dyalog APL/W
	AP119/PC	poa	TCP/IP interface for Dyalog APL/W
	FACS	poa	EMMA-like interface to DB2 or ODBC databases
	TOPR	poa	APL Code and Application Management for Dyalog APL/W
Mercia	LOGOL 92	poa	Logistics management system for 386/486 & RISC computers. Sales Forecasting, Inventory Management, Master Scheduling, Distribution Requirements Planning, Sales & Operations Planning.
	TWIGS	poa	A modular library of tools to teach and explore state-of-the-art materials management concepts. Developed by R.G. Brown.

RE Time Tracker Oy	UIT/W	poa	TMT-Team Oy's User Interface Toolkit for APL*PLUS II and PLUS III under Windows. Comprehensive spreadsheets, replicated fields, special field types, etc.
	DB+	poa	TMT-Team Oy's database interface for APL*PLUS II & PLUS III under Windows. Interfaces to almost twenty different databases.
Soliton Associates	LOGOS	poa	Application Development Environment
	MAILBOX	poa	Electronic Mail
	VIEWPOINT	poa	Report generator with interfaces to DB2 and MVS data
UNIWARE (for mainframe)	STSC's ENHANCEMENTS	poa	Quad-functions & nested arrays for IBM VSAPL
	STSC's SHAREFILE	poa	component files for IBM VSAPL and for IBM APL2
	TOOLS & UTILITIES	poa	including FILEPRINT, FILESORT, FILECONVERT FILEMANAGER(EMMA) STSC's database package
	EXECUCALC	poa	Mainframe spreadsheet compatible with VISICALC and part of LOTUS 1-2-3 under VSAPL(VM or TSO)
(for APL*PLUS/PC)	APL Debugger 2.1	FF1950 FF9750	A visual APL debugger to help develop applications (site license)
	Menus 3.0	FF2450 FF12250	Complete set of hierarchical menu utilities (site license)
	ETATGEN 2.0	FF1950 FF9750	Page layout report generator (site license)
	UNITAB 2.0	FF4550 FF22750	An APL*PLUS spreadsheet-like data entry and validation system (site license)
	UNIASM 3.0 (site license)	FF4950	Assembler utilities to speed up APL*PLUS/PC applications
	UNISTAT 5.1	FF2900	Data analysis add-on module for Statgraphics
(for APL*PLUS II)	UNIWARE Toolkit II 4.1	FF39000	(site license only). Relational database system and complete set of utilities for APL*PLUS II development
	APL Debugger II 2.1	FF2950 FF14750	A visual APL debugger to help develop applications (site license)
	Menus II 4.0	FF3950 FF19750	Complete set of hierarchical mouse-driven menu utilities (site license)
	ETATGEN II 2.0	FF2950 FF14750	Page layout report generator (site license)
	UNITAB II 2.0	FF6950 FF34750	An APL*PLUS spreadsheet-like data entry and validation system (site license)
	UNISTAT Plus 5.2	FF4300	Data analysis add-on module for Statgraphics
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.
Zark	APL Tutor (PC)	$299	APL computer-based training. Available for APL*PLUS PC & APL*PLUS II. Demo disk $10.
	APL Tutor (MF)	$5000	Mainframe version.
	Zark ACE	$99	APL continuing education. APL tutor news and hotline phone support.
	APL Advanced Techniques....	$59.95	488pp. book, (ISBN 0-9619067-07) including 2-disk set of utility functions (APL*PLUS PC format).
	Communications	$200 pc, $500 mf	Move workspaces or files between APL environments.

APL CONSULTANCY

COMPANY	PRODUCT	PRICES(£)	DETAILS
Active Workspace	APL Programming	poa	Short or long-term consultant available.

Adfee	Consultancy	poa	Development, maintenance, conversion, migration, documentation, of APL products in all APL environments
Andrews	Consultancy	poa	APL programming and analysis, specialises in tree-processing algorithms.
APL People	Consultancy	poa	Consultants available at all levels. Expertise in APL system design, project management, prototyping, financial applications, decision support systems, MIS, links to non-APL systems, documentation, etc.
Bloomsbury Software	Consultancy	300-750+VAT	
Camacho	Consultancy	poa	Manuals; feasibility reports and estimates; analysis and programming; APL and MS Windows applications; Sharp, ISI APL, APL*PLUS, APL2/PC and other APLs spoken. Fixed price systems a speciality
Ray Cannon	Consultancy	poa	APL, C, Assembler, Windows, Graphics: PC and mainframe
Causeway	Consultancy and Training	poa	Management and upkeep of the Causeway development environment for individuals and large APL sites. On-site training for Causeway/Dyalog and Causeway/Plus III
Paul Chapman	Consultancy	poa	24-hr programmer; APL, C, Assembler, Graphics; PC, mini, mainframe and network.
David Crossley	Consultancy	poa	Broad experience in many APL environments
Peter Cyriax	Consultancy	100-150 120-200 160-300	Junior Consultant Consultant Senior Consultant
Dogon Research	Consultancy	poa	APL Systems consultancy, design, implementation, support, documentation and maintenance. All dialects with special emphasis on APL2 and Dyalog APL/W.
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
Evestic AB	Consultancy	poa	Excellent track record from 10+ years of APL applications in banking, insurance, and education services. All dialects, platforms and project phases. SQL expertise.
General Software	Consultancy	from 120	
Greymantle Assoc	Consulting	poa	Company reporting, business graphics, Windows applications with Dyalog APL/W.
H.M.W.	Consultancy	poa	System design consultancy, programming. HMW specialize in banking and prototyping work.
Michael Hughes	Consultancy	poa	Consultant with 10+ years experience with various APL interpreters and C.
IAC/Human Interfaces	Consultancy	350	APL on Macintosh & PC. HCI design. VDU ergonomics: EC/Health & Safety compliance.
	Documentation	100-200	On-line assistance, product demos & mock-ups, manual writing; foreign language software localization.
	Training	poa	Using I-APL for courseware & distance learning materials; Mac programming in C, APL & HyperCard.
INFOSTROY	Consultancy	poa	APL*PLUS & Windows consultancy. Porting of software written in C into APL*PLUS.
Insight Systems	Consultancy	poa	Experts in APL conversions between any combination of: APL*PLUS, APL2, Dyalog APL and Sharp APL. We are also experienced right-sizers, comfortable with networks and relational databases (that also means when NOT to use SQL) and client/server development in APL, C and Visual Basic.
Intelligent Programs	Consultancy	poa	Systems development, enhancements, support.
	Documentation	poa	Preparation of new manuals, rewriting of existing materials.
	Training	poa	Training for APL experts through to non-technical system users.

JAD Software	Consultancy	poa	Systems design and development, project management, technical manuals, financial and actuarial expertise in APL
Kestrel	Consultancy	poa	All APLs, all environments. Design, analysis, coding, maintenance, documentation, training, interfacing.
Lingo Allegro USA	Consultancy	poa	General APL consulting: Migration and Downsizing; Performance Tuning.
MasterWork Software	Consultancy	poa	Consultancy
MicroAPL	Consultancy	poa	Technical & applications consultancy.
Ellis Morgan	Consultancy	250-500	Business Forecasting & APL Systems.
Optima	Consultancy	poa	A range of consultants with 3-15 yrs APL PC and mf experience.
QB On-Line	Consultancy	350	Specialising in Banking, Financial & Planning Systems.
RE Time Tracker Oy	Consultancy	poa	Specialised in comprehensive APL Windows user interfaces, APL Multimedia, APL to API level interfacing for Windows, Windows applications, DLLs & databases.
Rex Swain	Consultancy	poa	Independent consultant, 20 years experience. Custom software development & training, PC and/or mainframe.
Rochester Group	Consultancy	poa	Specialise in MIS using Sharp APL
Snake Island Research Inc	Consultancy	poa	APL interpreter and compiler enhancements, intrinsic functions, performance consulting. APL compiler for serial and parallel systems now under test.
Strand Software	Consultancy	poa	Advice on migrating to and from all flavours of APL and hardware platforms. Full-screen interface implementation, APL utilities, benchmarking, efficiency analysis, actuarial software, system development tools, valuation, pricing and modelling systems.
Sykes Systems Inc	Consultancy	poa	Complete APL services specialising in audit, optimisation and conversion of APL systems. Excellent design skills. All dialects and platforms. 17-23 years experience.
Uniware	Consultancy (Senior)	FF/day 5000	Consultancy from people with at least 8 years APL experience.
	Consultancy (Senior)	FF/day 7500	Advice and training in Windows programming with APL*PLUS II
	Training	FF10000	5-day class on Windows programming with PLUS II version 4.0
Wickliffe Computer	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	Employees placed at all levels.
Bloomsbury Software	Training	poa	Contact the company for details.
ComLog	Comic-Logger	$25.95+p&p	APL*PLUS II comic-book inventory system. Shareware version available on America OnLine.
HMW	Employment	poa	Contractors and permanent employees placed.
HRH Systems	APL lessons		On-screen interactive APL lessons for APL*PLUS, TryAPL2, Sharp and I-APL — in English or French.
	The BBS\APL:	$24 p.a.	703-528-7617, 1200-14400b, N-8-1, 24 hours. APL educational material is downloadable free. An additional 30 megs of APL software for APL*PLUS, PLUS II, IBM, Sharp & I-APL is available to subscribers (cost is $24/yr). Selection available on disk for $15 post-paid. Free on-disk catalogue.
I-APL Ltd	An APL Tutorial	3	45pp by Alvord & Thomson
	An Encyclopaedia of APL (2d Ed)	6	228pp by Helzer
	APL in Social Studies	3	36pp by Traberman
	I-APL Instruction Manual (2d Ed)	3	55pp by Camacho & Ziemann

	APL Programs for the Mathematics Classroom (Springer-Verlag)	16	185pp by Thomson
	Programming in J	10	75pp by Ken Iverson
	Arithmetic	12	118pp by Ken Iverson
	Tangible math	8	36pp by Ken Iverson
	Sharp APL Reference Manual	42	349pp by Berry
	APL Press Books	poa	A comprehensive selection of early APL literature
	Please note there is a packing charge of £3 per order		
Kestrel	Employment	poa	Permanent and contract, home and abroad. From individual placement to supply of complete project teams.
	Software Library	poa	Low-cost software distribution service; call for details.
Renaissance Data Systems	Booksellers		The widest range of APL books available anywhere. See Vector advertisements.
Soliton Associates	MVSLINK	poa	Interface from Sharp APL (Unix & MVS) to non-APL data and software in the MVS environment.
	SSQL	poa	High-performance DB2 interface for Sharp APL (Unix and MVS).

OVERSEAS ASSOCIATIONS

GROUP	LOCATION	JOURNAL	OTHER SERVICES	Ann.Sub.
ACM/SIGAPL	International	Quote Quad		
APL Bay Area	USA N. California	APLBUG	Monthly Meetings (2nd Monday)	$15
APL Club Austria	Austria	-	Quarterly Meetings	200AS(indiv), 1000AS(corp)
APL Club Germany	Germany	APL Journal	Semi-annual meetings	DM60
APL Interest Group	South Africa	-	-	
Ass. Francophone pour la promotion d'APL	France	Les Nouvelles d'APL		
BACUS	Belgium	APL-CAM	Conferences & Seminars	£18 ($30)
Chicago SIG	Chicago			
Hartford Group	Hartford, Connecticut, USA			
Capital PCUG	Washington, D.C.	Monitor	Monthly meetings, occasional classes	free
Danish SIG	Denmark			
Dutch APL Assoc.	Holland	-	Mini-congress, APL ShareWare Initiative	
FinnAPL	Helsinki, Finland	FinnAPL Newsletter	Seminars on APL	100FIM(private), 30(student), 1000 (Co)
Japan APL Assoc.	Tokyo, Japan			
Melbourne APLUG	Melbourne, Australia		Quarterly meetings	free
New York SIG	New York, USA			
Potomac SIG	Washington DC, USA		Free monthly meetings	
Rochester APL	Rochester, New York			
Rome/Italy SIG	Roma, taly			
SE APL Users Grp	Atlanta, Georgia	SEAPL Newsletter		
SOCAL	Southern California		Seminars	$15 ($5 students)
SovAPL	Obninsk, Russia			
SwedAPL	Sweden	SwedAPL Nytt	Semi-annual meetings, seminars	SEK 75
SWAPL	Texas, USA	SWAPL		$18
Swiss APL (SAUG)	Bern	Part of Qtly SI-Info		SF60 (SI) + SF20 (SAUG)
Sydney APLUG	Sydney, Australia	Epsilon	Monthly Meetings	
Toronto SIG	Toronto, Canada	Gimme Arrays!	Monthly Meetings, APL skills database, J SIG, Toronto Toolkit	$25

VENDOR ADDRESSES

COMPANY	CONTACT	ADDRESS & TELEPHONE No.
ACM/SIGAPL	Donna Baglio	ACM, 1515 Broadway, New York, NY 10036 USA Tel:+1 (212) 626-0606 Email: baglio@acm.org
Active Workspace Ltd	Ross D Ranson	Moulsham Mill Centre, Parkway, Chelmsford, Essex, CM2 7PX, UK. Tel: 01245-496647; Fax: 01245-496646.
Adaptable Systems	Lois & Richard Hill	49 First Street, Black Rock 3193, Australia. Tel: +61 3 589 5578 Fax: +61 3 589 3220
Adaytum Systems		13 Great George Street, BRISTOL BS1 5RR UK Tel: 0117-921 5555
Adfee	Bernard Smoor	Dorpsstraat 50, 4128 BZ Lexmond, Netherlands. Tel +31-3474-2337, Fax: +31-3474-2342
Andrews	Dr Anne D Wilson	23, The Green, Acomb, YORK YO2 5LL, UK Tel: 01904-792670
APL-385	Adrian Smith	Brook House, Gilling East, York YO6 4JJ UK. Tel: 01439-788385 Fax: 01439-788194 Email: 100331.644@compuserve.com
APL Bay Area Users Group APLBUG	Lewis H. Robinson (Sec)	1100 Gough St, Apt 14A, San Francisco, CA 94109, USA Tel: +1 (415) 928-2058 Email: frgp21a@prodigy.com
APL Club Austria	Erich Gall	IBM Österreich, Obere Donaustrasse 95, A-1020 Wien, Austria
APL Club Germany	Dieter Lattermann	Rheinstrasse 23, D-69190 Walldorf, Germany. Tel: +49 6227-63469 Compuserve: 100332,1461
The APL Group Inc	Stuart Sawabini	644 Danbury Road, WILTON, CT 06897 USA. Tel: +1 (203) 762-3933 Fax: +1 (203) 762-2108
APL Interest Group, South Africa	Mike Montgomery	Private Bag X11, Rivonia 2128, South Africa Tel: +27 (11) 803-7200 Fax: +27 (11) 803-9134 Email: mikemont@spl.co.za
APL People / Software	Jill Moss	The Old Malthouse, Clarence St, BATH, BA1 5NS UK. Tel: 01225-462602
Association Francophone pour la promotion d'APL	Dr. Gérard Langlet	SCM, C.E. Saclay, F-91191-Gif sur Yvette, France. Fax:+33 1 69-08-79-63
Atlantis Software	Arthur Whitney	1105 Harker Avenue, Palo Alto, CA 94301 USA
BACUS	Joseph de Kerf	Rooinberg 72, B-2570 Duffel, Belgium. Tel: +32 15 31 47 24
Beautiful Systems, Inc.	Jim Goff	308 Old York Road, Suite 5, Jenkintown, PA 19046, USA Tel: +1 (215) 886-2636; Fax: +1 (215) 886-4888
The Bloomsbury Software Co Ltd	Peter Day	3-6 Alfred Place, Bloomsbury, London WC1E 7EB UK. Tel: 0171-436 9481; Fax: 0171-436 0524; CompuServe: 100010,1467
Camacho	Anthony Camacho	11 Auburn Road, Redland, Bristol BS6 6LS UK. Tel: 0117-9730036. email: acamacho@cix.compulink.co.uk Reutemet (Sharp): ACAM
Ray Cannon		21 Woodbridge Rd, Blackwater, Camberley, Surrey GU17 0BS UK Tel: 01252-874697 Email: 100430.740@compuserve.com
Paul Chapman		51B Lambs Conduit Street, London WC1N 3NB UK. Tel: 0171-404 5401. Compuserve: 100343,3210
Causeway Graphical Systems Ltd	Adrian Smith, Duncan Pearson	5 The Maltings, Castlegate, MALTON, North Yorks YO17 0DP UK Tel: 01653-696760 Fax: 01653-697719 Compuserve: 100265,1564
Chicago SIG	Larry Mysz	836 Highland Drive, Chicago Heights, IL 60411 USA C'serve:73040,3032
Cinerea AB	Rolf Kornemark	Skyttegatan 25, S-193 00 Sigtuna, Sweden.
CODEWORK	Mauro Guazzo	Corso Cairoli 32, 10123 Torino, Italy. Tel: +39 11 885168 Fax: +39 11 812 2652
ComLog Software	Jeff Pedneau	PO Box 5570, Derwood, MD 20855 USA Tel: +1 (301) 990-7063 Email: jeff@softmed.com
CPCUG	Lynne Startz	Capital PC User Group, 51 Monroe Street, Suite PE-2, Rockville, Maryland 20850-2421, USA. Tel: +1 (301) 762-9372 Fax: (301) 762-9375.
David Crossley		187 Le Tour du Pont, Quartier Le Mourre, 84210 ST DIDIER, France Tel: +33 90-66-08-87
CYBEX AB	Lars Wentzel	Gruvgatan 35B, S-421 30 V. Frölunda, Sweden. Tel: +46 31-45 37 40. Fax: +46 31-45 24 23.
Peter Cyrlax Systems	Peter Cyrlax	22 Hereford Road, London W2 4AA UK. Tel: 0171-229 5344
Danish User Group	Per Gjerlof	Email: gjerper@inet.uni-c.dk

Datatrade Ltd.	Ian Tomlin	1 & 2 Sterling Business Park, Salthouse Road, Brackmills, Northampton, NN4 0EX UK. Tel: 01604-760241
Dogon Research	Dick Bowman	2 Dean Gardens, London E17 3QP UK Tel: 0181-520 6334 Email:bowman@apl.demon.co.uk
Dutch APL Association	Bernard Smoor (Sec)	Postbus 1341, 3430BH Nieuwegein, Netherlands. Tel: +31 3474-2337
Dyadic Systems Ltd.	Peter Donnelly	Riverside View, Basing Road, Old Basing, Basingstoke, Hants RG24 0AL UK. Tel: 01256-811125 Fax: 01256-811130
E & S Associates	Frank Evans	19 Homesdale Road, Orpington, Kent BR5 1JS UK. Tel: 01689-824741
Evestic AB	Olle Evero	Bertellusvagen 12A, S-146 38 Tullinge, Sweden Tel&Fax: +46 778 4410
FinnAPL		Suomen APL-Yhdistys RY, FinnAPL RF, PL 1005, 00101 Helsinki 10, Finland
General Software Ltd	M.E. Martin	22 Russell Road, Northholt, Middx, UB5 4QS UK. Tel: 0181-864 9537
Greymantle Associates	George MacLeod	Bartrum House, Ravens Lane, Berkhamsted, Herts, HP24 2DY UK Tel: 01442-878065 Email: 100412,1305@compuserve.com
Hartford CT Group	Bob Pomeroy	Mass Mutual Life, 1295 State St, Maildrop F465, Springfield, MA 01111 USA Tel: +1 (413) 788-8411x2838
H.M.W.Trading Systems Ltd	Stan Wilkinson	Hamilton House, 1 Temple Avenue, Victoria Embankment, London EC4Y 0HA UK. Tel: 0171-353 8900; Fax: 0171-353 3325; Email:100020.2632@ compuserve.com
HRH Systems	Dick Holt	3802 N Richmond St, Suite 271, Arlington, VA 22207 USA Tel: +1 (703) 528-7624; Email: dick.holt@acm.org
Michael Hughes		28 Rushton Road, Wilbarston, Market Harborough, Leics. LE16 8QL UK. Tel: 01536-770998
IAC/Human Interfaces	Ian A. Clark	9 Hill End, Frosterley, Bishop Auckland, Co. Durham DL13 2SX UK Tel: 01388-527190. Compuserve: 100021,3073
I-APL Ltd	Anthony Camacho (for queries, order forms)	11 Auburn Road, Redland, Bristol BS6 6LS UK. Tel: 0117-9760036 email: acamacho@cix.compulink.co.uk Reutemet (Sharp): ACAM
	J C Business Services (for pre-paid orders only)	56 The Crescent, Milton, Weston-super-Mare, Avon, BS22 8DU UK
IBM APL Products	Nancy Wheeler	APL Products, IBM Santa Teresa, Dept M46/D12, 555 Bailey Avenue, San Jose CA 95141, USA. Tel: +1 (408) 463-APL2 (=2752) Fax: +1 (408) 463-4488 Email: APL2@vnet.ibm.com Cserve: GO IBMAPL2
Impetus Ltd	Cedric Heddle	Rusper, Sandy Lane, Ivy Hatch, SEVENOAKS, Kent TN15 0PD UK Tel: 01732-885126
INFOSTROY	Alexei Miroshnikov	3 S. Tulenin Lane, St. Petersburg 191186 Russia. Tel:+7 812-312-2673 Fax:+7 812-311-2184 Email:aim@infostroy.spb.su
Insight Systems ApS	Morten Kromberg	Nordre Strandvej 119A, DK-3150 Hellebæk, Denmark Tel:+45 42 10 70 22 Fax: +45 42 10 75 74 Email: insight@inet.uni-c.dk
Intelligent Programs Ltd	Mike Bucknall	9 Gun Wharf, 130 Wapping High St, London E1 9NH Tel: 0171-265 1120
Interprocess Systems Inc.	Stella Chamberlain	11660 Alpharetta Highway, Suite 455, Roswell, Georgia 30076, USA Tel: +1 (404) 410-1700. Fax: +1 (404) 410-1773 Cserve: 70373,2676
Iverson Software Inc.	Eric Iverson	33 Major Street, Toronto, Ontario, Canada M5S 2K9 Tel: +1 (416) 925-6096; Fax: +1 (416) 488-7559
JAD Software	David Crossley	580 Eyer Drive, #81 Pickering, Ontario, Canada L1W 3B7 Tel: +1 (905) 837-1895 Fax: +1 (905) 831-5172
Japan APL Association		23-2-302 Hiromichi, Adachi-ku, Tokyo 120, Japan
Kestrel Consulting	Mark Harris	Business & Technology Centre, Bessemer Drive, Stevenage, Herts SG1 2DX UK. Tel: 01438-310155 Fax: 01438-310131
Lingo Allegro USA Inc.	Victoria H. Fil	113 McHenry Road, Suite 161, Buffalo Grove, IL 60089 USA Tel:+1 (708) 459-7529 Fax:+1 (708) 459-8501 Email: info@lingo.com
MasterWork Software Ltd	Fraser Jackson	PO Box 56-036, Tawa, Wellington, New Zealand. Tel and Fax: +64 (4) 232-4440 Email: 100242.2535@compuserve.com
Melbourne APL Group	Harvey Davies	CSIRO Div Atm Res, Private Bag No.1, Mordialloc, Victoria 3195, Australia Tel: +61 3 586 7574 Fax: +61 3 586 7600 Email: hld@dar.csiro.au
Mercia Software Ltd.	Gareth Brentnall	Holt Court North, Heneage Street West, Aston Science Park, Birmingham B7 4AX UK. Tel: 0121-359 5096. Fax: 0121-359 0375
MicroAPL Ltd.	David Eastwood	South Bank Technopark, 90 London Road, LONDON SE1 6LN UK Tel: 0171-922 8866 Fax: 0171-928 1006 Email: MicroAPL@microapl.demon.co.uk

Ellis Morgan	Ellis Morgan	Myrtle Farm, Winchester Road, Stroud, Petersfield, Hants UK. Tel: 01730-263843
New York SIG APL	Nestor Nelson	PO Box 138, NY 10185-0002, USA
Optima Systems Ltd	Paul Grosvenor	Airport House, Purley Way, Croydon, Surrey CR0 0XY UK. Tel: 0181-781 1812 Fax: 0181-781 1999
Potomac APL SIG	John Martin	51 Monroe Street, Plaza East Two, Rockville MA 20850-2421 USA Tel: +1 (301) 762-9372 Fax: +1 (301) 762-9375 Email:jam@acm.org
QB On-Line Systems	Philip Bulmer	5 Surrey House, Portsmouth Rd., Camberley, Surrey, GU15 1LB UK. Tel: 01276-855880 Fax: 01276-855301
Renaissance Data Systems	Ed Shaw	PO Box 421, Georgetown, CT 06982, USA. Tel: +1 (212) 864-3078
RE Time Tracker Oy	Richard Eller	PO Box 363, FIN-00101 Helsinki, Finland. Tel: +358-0-400 2777
Rochester APL	Gary Dennis	Soliton Associates, 1100 University Avenue, Rochester, NY 14607 USA Email: gsd@ipsalab.tor.soliton.com
The Rochester Group Inc.	Robert Pullman	50 S.Union St., Rochester NY 14607-1828, USA. Tel: +1 (716) 454-4360. Fax: +1 (716) 454-5430
Rome/Italy SIG	Mario Sacco	Casella Postale 14343, 00100-Roma Trullo, Italy Email: marsac@vnet.ibm.com
SE APL Users Group	John Manges	991 Creekdale Drive, Clarkston, GA30021 USA
Shandell Systems Ltd.	Maurice Shanahan	Chiltern House, High Street, Chalfont St. Giles, Bucks HP8 4QH UK.
Snake Island Research Inc	Bob Bernecky	18 Fifth Street, Ward's Island, Toronto, Ontario M5J 2B9 Canada Tel: +1 (416) 203-0854 Fax: +1 (416) 203-6999 Email: bernecky@eecg.toronto.edu
SOCAL (South California)	Roy Sykes Jr	Sykes Systems Inc, 4649 Willens Ave, Woodland Hills, CA 91364-3812 USA Tel: +1 (818) 222-2759 Fax: +1 (818) 222-9250
Soliton Associates	Laurie Howard	Soliton Associates Ltd, Groot Blankenberg 53, 1082 AC Amsterdam, Netherlands Tel: +31 20 646 4475 Fax: +31 20 644 1206 Email:ljh@soliton.com
SovAPL	Alexander Skomorokhov	PO Box 5061, Obninsk-5, Kaluga Region, Russia Email:askom@apl2.obninsk.su
Strand Software Inc	Anne Faust	19235 Covington Court, Shorewood MN 55331 USA Tel: +1 (612) 470-7345 Email: amfaust@aol.com
Rex Swain	Rex Swain	8 South Street, Washington, CT 06793 USA. Tel: +1 (860) 868-0131 Fax: +1 (860) 868-9970 Email: rswain@ix.netcom.com CompuServe: 70062,2303
SWAPL	Stuart Yarus	PO Box 210367, Bedford, Texas 76095, USA Tel: +1 (817) 577-0165 Compuserve: 73700,2545
SwedAPL	Christer Ulfhielm	Novator Consulting Group AB, Svärdvägen 11C, S-182 33 Danderyd Sweden Tel: +46 8 622 63 50 Fax: +46 8 622 63 51 CServe: 100341,404
Swiss APL User Group		Swiss APL User Group, CH-3001, Bern 1, Switzerland Email: si@ifi.unizh.ch
Sydney APLUG	Rob Hodgkinson	PO Box 1511, Macquarie Centre, NSW 2113, Australia Tel:+61 2 257 5313
Sykes Systems Inc	Roy Sykes Jr	4649 Willens Ave., Woodland Hills, CA 91364, USA Tel: +1 (818) 222-2759 Fax: +1 (818) 222-9250
Toronto SIG	Ben Best	PO Box 55, Adelaide St. Post Office, Toronto Ontario M5C 2J5, Canada Email: benbest@io.org
Uniware	Eric Lescasse	Tour Neptune, Cedex 20, 92086 Paris la Defense 1, France. Tel: +33 (1) 47-78-78-00. Fax: +33 (1) 40-90-04-11
Wickliffe Computer Ltd	Nick Telfer	76 Victoria Rd, Whitehaven, Cumbria, CA28 6JD UK. Tel: 01946-692588
Warwick University	Prof. Jeff Harrison	Dept of Statistics, University of Warwick, Coventry, CV4 7AL UK Tel: 01203-523369
Zark Incorporated	Gary A. Bergquist	23 Ketchbrook Lane, Ellington CT 06029, USA. Tel: +1 (203) 872-7806

Dyalog APL for Windows 95

first impressions, by Adrian Smith

Disclaimer

Dyalog for Windows 95 arrived at the weekend, and is still not fully documented, so I cannot claim that this is in any way a full and fair evaluation. I have spent most of an afternoon playing with it – that is all.

Background

This is a full 32-bit native Windows program, and uses the Win32 dynamic link libraries, rather than the old 16-bit code. On the surface there is little visible change, but it loads quickly and offers easy access to all the new controls that form part of the '95' look and feel. The only conversion required is to ⎕*NA* calls which must be changed to use the new 32-bit libraries. Here you may need to do some research, for example I have yet to find the equivalent call to *SndPlaySnd* (in `mmsystem.dll`) which I have been using to annoy Duncan by making silly noises from within APL.

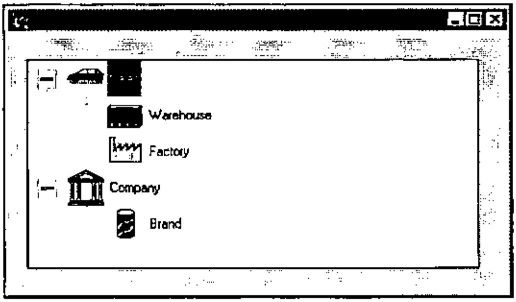
A major step towards Windows 95 integration is the addition of a *DropFiles* event to Forms and many of the 'List box' objects. The basic interface of Windows 95 is very much more Mac-ish, and users will expect to scatter directories (sorry, *folders*) all over the screen (sorry, *Desktop*) and be able to drop them on to application short-cuts or running applications. If you set the *AcceptFiles* property of your form to 1, it reports the name of the dropped file for you, which is just what you need to go and open it (having checked the extension to ensure it is one of yours!) without the user ever touching a menu or toolbar button.

The rest of this short review concentrates on the new Windows controls, hopefully giving you some clue as to where you might find them useful.

Some of the New Controls

Here are a few of the things I was able to turn over this afternoon. All of them worked first time, and I only blew the interpreter out of the water once. Even here, I can report a major improvement in behaviour – it shut down without causing any damage to Windows, and I was able to restart Dyalog immediately.

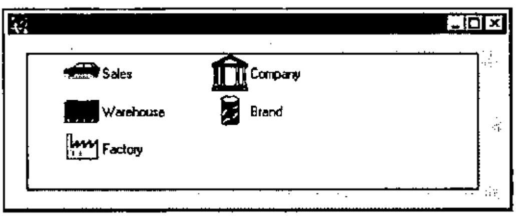
The Tree Viewer



This supersedes the `msoutlin` control from Visual Basic, but is used in a very similar way to build indented lists which the user can roll up and uncurl as required. As you can see, it is very easy indeed to put pictures into it, and you can also have more than one item at the top level (an annoying limitation of the old tree). I tried it with some very big lists (easily enough to blast the old .VBX to kingdom come) and some ludicrous indents (up to 1000, after which the control has had enough and keels over, but fair enough I think).

In summary, it looks good, is dead easy to use and I can think of lots of places I could put it to work. 10/10 to Bill Gates for this one.

The List Viewer



Here is what it took to make the pictures (also used in the tree shown above) and then create a window to manage them as a list:

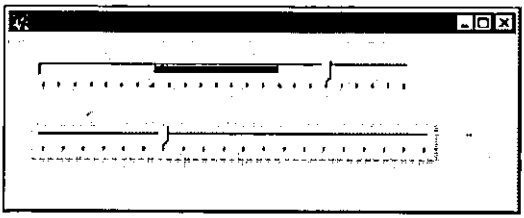
```
'il' ⎕wc 'ImageList' (32 32)
'il.' ⎕wc 'Icon' 'pics\ou-so'
'il.' ⎕wc 'Icon' 'pics\ou-whse'
'il.' ⎕wc 'Icon' 'pics\ou-plant'
'il.' ⎕wc 'Icon' 'pics\ou-co'
'il.' ⎕wc 'Icon' 'pics\ou-dv'
'ff.lv'⎕WC'ListView'('Posn' 16 16)(120 400)
'ff.lv'⎕WS'Items' ('Sales' 'Warehouse' ...)
'ff.lv'⎕WS ('Imagelist' 'il')('ImageIndex' 0 1 2 3 4)
'ff.lv'⎕WS'View' 'List'
```

This strings a bunch of icons into the new *ImageList* object and then sets this list as the property of the *ListView*. You can re-use the icons as many times as you like, as you specify which picture to associate with each item by a separate *ImageIndex* property. I was in ⎕*IO*=0 here, by the way.

Once you have your list, you can get all the different views that Windows itself provides from Explorer. If you choose to tabulate the items, the object will let you set the headings, column widths and so on. The user can also drag the items around, and (if you set `AutoArrange` to `1`) the object will line them all up nicely in the new sequence.

In summary, some imagination will be needed to make good use of this one, not to mention an artistic streak to make decent icons (both 32 by 32 and 16 by 16 sizes required). Let's give Bill $^{8}/_{10}$ for effort.

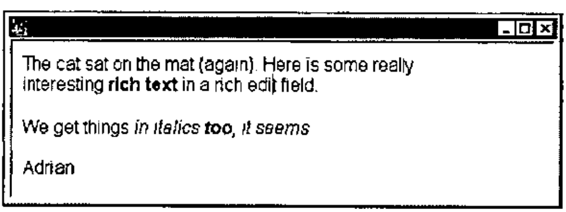
The TrackBar



Two basic styles, either of which can be used horizontally or vertically (by setting the `Hscroll` and `Vscroll` properties - yuk) to set the volume on your sound card, record the results of a questionnaire (the wishy-washy qualitative kind), play with the fitness factors for your Genetic Algorithms, and so on. The range and tick values can all be set, and the range is no longer limited to values from 1 upward, *mirabile dictu.*

The major improvement here (apart from the visuals) is that it gives you a stream of events as it tracks, so you can steer the aeroplane, zoom the view, or rotate the 3D barchart ***as the user pulls the slider,*** rather than waiting for him to drop it. PLUS III already had this on scroll bars - but this is better for the job it does. In summary, another winner. $^{10}/_{10}$ again.

The Rich Edit Field



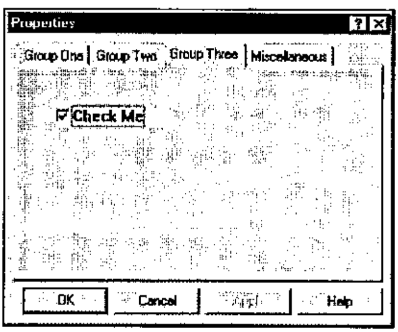
This is going to take some getting used to! It is more or less a complete word processor in a box, with font and formatting properties which apply to the *currently selected text,* rather than the whole content of the object. It will read and display existing .RTF files (such as those saved by Word) or you can save text (such as the above) to file, and read it in Word with no problems. More interesting from a programming angle - you can get at the `RtfText` property directly. This is always a simple vector and has all the formatting codes embedded. I can see that for applications which must build and print form letters (with the punter's name in italics and the special offer in a fancy bold font), this could be a god-send. I think I had better reserve judgement on this one, but 8/10 or better is likely. Approach it cautiously, with manual in hand.

Property Page and Property Sheet

This one is lovely! Having tangled with the tabbed subforms in the current Dyalog release, it is wonderful to throw out all that horrible stuff with TabBtns (which never quite fitted right as soon as you changed the defaults) and just let Windows do it for you.

The Property Sheet is a top-level object, which comes ready supplied with buttons for OK, Cancel, Apply and (optionally) Help. As you add Property Pages to it, they appear with their titles in the tabs, correctly aligned and ready to use. You then simply add objects to each page, and everything else is done for you.

```
'ps' ⎕wc 'PropertySheet' 'Properties' (0 0)(120 400)
    ('coord' 'pixel')
'ps.pp1' ⎕wc 'PropertyPage' 'Group One'
'ps.pp2' ⎕wc 'PropertyPage' 'Group Two'
'ps.pp3' ⎕wc 'PropertyPage' 'Group Three'
'ps.pp4' ⎕wc 'PropertyPage' 'Miscellaneous'
'ps.pp3.ck'⎕WC'Button' 'Check Me' (32 32)(24 120)'Check'
```



If I could give Bill 11/10, I would. A lot of messy Causeway code goes out of the window, replaced by a 'Properties' object which does the same job faster and better.

Conclusions

I have not had time for a decent go at the spinner or progress bar – again both replace existing Causeway objects but will be quicker to build and more responsive. If you are starting from raw Dyalog, the Spinner will save you a lot of time and some quite subtle Gui programming.

Basically, this all came straight off CompuServe, we unzipped it and it worked. The new objects are excellent, but don't expect any startling change in the rest of the product until later in the preview cycle. An essential upgrade for the serious Windows developer.

RECENT MEETINGS: DDE Workshop

Presenter's Notes by Duncan Pearson

In this article I will not cover all of the material from the July talk but will concentrate on those areas that have not already been covered in Vector articles. I will explain some of the terms used in talking about DDE and how they relate to the implementation of DDE in Dyalog APL/W and APL*Plus III. All of the previous Vector articles on this subject have covered DDE from and to Dyalog APL/W and so I will concentrate on DDE from the point of view of APL*Plus III. The examples that I use will be from the July talk, but updated where appropriate to use Excel 5 or above. I will show two APL workspaces sharing a "common" edit field, an APL workspace sharing data with an Excel spreadsheet and finally a small application in which a Word document can contain fields linked to live data in APL via an intermediate Excel spreadsheet.

Some Terms Explained

DDE was designed to allow applications to share their data and so the terms used in DDE are really just a common way of specifying what data one application wants to share with another. The three main elements of this specification are "Application", "Topic" and "Item".

- **Application**. This is the name of a Windows application. In general it is the name of the .EXE file used to start the application, although there are exceptions.
- **Topic**. In document-centred applications this is generally the name of the document from which the data is to be taken. In the case of Dyalog APL/W it is the name of the workspace, in the case of APL*Plus it is the name of a form.
- **Item**. This specifies precisely what data is required. In Dyalog APL/W it is the name of a variable and in APL*Plus III it is the name of a label or edit field on a form. In Excel it is a cell range specification or a defined range name and in Word it is a bookmark defining a vector of text.

Differences in Approach

In the meanings attached by the different interpreters to the DDE terms we see the main difference of approach. In Dyalog APL/W it is the *variables* of the workspace that are linked with other applications. In APL*Plus III it is the *text elements* of the GUI that the APL application has built that are linked. The means by which the interpreter notifies the application of changes to the watched data is also different. In Dyalog APL/W when a linked variable changes as a result of activity in another application then a DDE event occurs on the root object. This message does not specify which variable has changed whereas in APL*Plus III each individual GUI element that changes will receive a "Changed" event which can result in the execution of code in the normal manner. This is more specific and with more than one element linked this distinction is important.

In the shared variable paradigm with which many APLers will be familiar, the processes on each side of the share are considered equal. This is not the case with DDE. The process requesting the share is considered the *"Client"* and the process that accepts the share and thus provides the data is considered the *"Server"*.

Setting Up a DDE Client

In APL*Plus III you link either a label or an edit field as the client of another application. Both the edit field and the label have specific properties where you can specify the application, topic and item of the data you wish to link.

If the label is called "form1.label1" then:

```
'form1.label1'⎕wi'ddeApplication' 'Excel'
'form1.label1'⎕wi'ddeTopic' '[book1]sheet1'
'form1.label1'⎕wi'ddeItem' 'R1C4:R3C4'
```

will define it as linked to a three-cell selection D1:D3 in sheet1 of book1 in Excel.

There are two ways in which the link can be made. If the link is "cold" then the client application is not told of changes to the data with which it is linked. It must request the data. If the link is "hot" then the server application should notify the client application whenever the data changes. Some applications, notably Excel, are a little over-zealous in the notification; if any cell in a sheet changes then applications linked to any of the cells in that sheet are notified, even if the linked cells have not changed at all. This problem will crop up later.

The following line defines the link as cold.

```
'form1.label1'⎕wi'ddeMode' 'cold'
```

It is at this point that APL*Plus III actually tries to initiate the link. Only now will you find out whether the names specified for the application, topic and item are correct. If not APL will generate an error:

```
⎕WI DDE ERROR: Unable to connect with ...
```

Finally we must request from Excel the data for the label. The `DdeRequest` method on the label does this and returns its success or failure:

```
      'form1.label1'⎕wi'DdeRequest'
1
```

Setting Up a DDE Server

If APL*Plus is the server, then the topic and item requested of it must specify a label or edit field. Thus the topic identifies the form and the item identifies the label or edit field. However they are not the names of the form and label. The names by which they are identified are given in the "ddeTopic" property of the form and the "ddeItem" property of the label or edit.

```
  'form2'⎕wi'ddeTopic' 'ServerForm'
```

will set "form2" up as a DDE server. APL will at this point register itself with Windows as a DDE server.

```
  'form2.edit1'⎕wi'ddeItem' 'ItemString'
```

will give "edit1" the DDE item name "ItemString". Now in a cell in Excel if you were to type

```
  =APLW|ServerForm!ItemString
```

the contents of the edit field would be copied into the cell, and any further changes to the edit field would be reflected in the cell. In Word the corresponding mechanism for inserting a DDE link is the DDEAUTO field. It takes as parameters the application, topic and server separated by spaces. It is again a hot link and will update automatically. Unfortunately Microsoft have removed DDEAUTO from the "Insert Field" dialogue box and so you have to insert the field manually.

An Example Application

At the BAA talk a small application was shown which demonstrates some of the possibilities for systems interacting via DDE. It was a combination of a *Word template*, an *Excel spreadsheet* and an *APL workspace*. The user would insert fields into a Word document that would be linked to data in the APL workspace. Associated with each expression would be a "natural language" APL expression that would evaluate to a statistic. The field could be moved around the document and formatted in any way desired. Furthermore if the data underlying the statistic changed then the field would be updated.

The Excel Spreadsheet

This is at the heart of the system and consists of three columns and one extra cell.

In the first column are the expressions to execute, in the second are the results and the third column displays the values in either the first or the second column depending on the value of the single cell which is Boolean (TRUE/FALSE). It is this third column to which the fields in the Word document are linked and this allows a facility in the document to display either the expressions or the results of them. A macro within the Word template sets the value of the single cell to TRUE in order to see the results and to FALSE to see the expressions. A name is defined for the range of the first column and one also for the second. This allows the APL system which is linked to the columns to manage without knowing how many expressions have been entered.

The APL System

The APL system consists of a form on which are a label and an edit field. The label is "hot-linked" as client to the expression column of the spreadsheet and the edit field is offered as a server item to which the second column of the spreadsheet is linked as described above. The following function sets up the form.

```
SetUp
⍝ Set up the server form to calculate the results of the expressions in an
⍝  Excel spreadsheet and to place them in a field accessible by Excel

⍝ Initialise the record of the previous contents of the expression cells
olddata←''

⍝ Reset the GUI
'#'⎕wi'Reset'

⍝ Create the form
⎕wself←'form'⎕wi'New' 'Form' ('where' 1 1 15 30)

⍝ Set its DDE topic. At this point APLW declares to Windows that it is a DDE
⍝  server.
⎕wi 'ddeTopic' 'Calculate'

⍝ Create a multi-line edit field and give it the DDE item name "results"
⎕wself←⎕wi ':res.New' 'Edit' ('style' 4)('where' 1 13 11 10)
⎕wi 'ddeItem' 'results'

⍝ Create a label and link it to the "expressions" range in the spreadsheet
⍝  "[book1]sheet1"
⎕wself←⎕wi ':expr.New' 'Label' ('where' 1 1 11 10)
⎕wi 'ddeApplication' 'excel'
⎕wi 'ddeTopic' '[book1]sheet1'
⎕wi 'ddeItem' 'expressions'
```

```
⍝ Make the link "hot" so that a change in excel will automatically reflect
⍝  in the label
⎕wi 'ddeMode' 'hot'

⍝ Set it to run the function "UpdateResults" whenever it changes
⎕wi 'onChange' 'UpdateResults'

⍝ Request Excel that it provide the data to populate the label
⎕wi 'DdeRequest'
```

When the label changes (as a result of a new expression being entered or an existing expression amended) then the `UpdateResults` is run. This calculates the results of the expressions, formats them and places them in the edit field.

```
UpdateResults;newdata
⍝ Calculate the results of the expressions in the caption of ⎕wself
⍝  and place them in the field "res" of the same form.
⍝ Called on a change to the label "expr"

⍝ Get the expressions
newdata←⎕wi'caption'

⍝ Excel sometimes notifies a change when none has occured so we must check
:if ~newdata≡olddata
    olddata←newdata

⍝ Convert the newline delimited text vector to nested form
    newdata←1↓¨(1,⎕tcnl=newdata)⎕penclose ' ',newdata

⍝ Execute each subvector and place the concatenation of the results
⍝  in the result field "res"
    ⎕wi ':res.text' (1↓∊⎕tcnl,¨⍕¨⍎¨newdata)
:endif
```

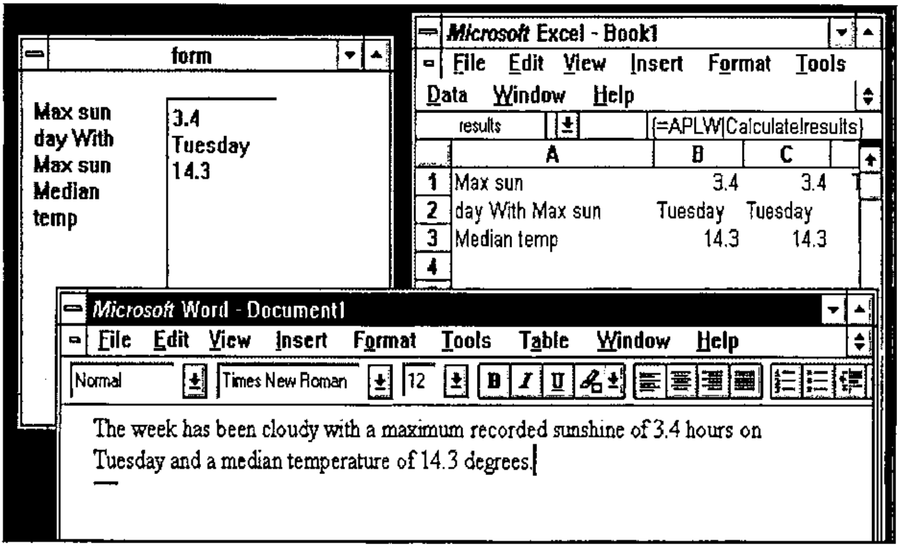
The only complication is the check to see that the data has in fact changed. Excel, as described above, notifies a change to all cells when any cell is changed; when we change the edit field and thereby the results column of the spreadsheet we are notified of a change in the expressions, causing us to update the edit field again ...

The functionality of the system is provided by functions and variables in the workspace so that a flexible query language is implemented.

The Word Template

This contains the macros that manage the contents of the expressions column in the spreadsheet and make sure that the names defined there cover the full range of expressions. There is also a macro to change between "result" and "expression" view by (again using DDE) poking a Boolean value into the cell D1.

The system is shown in action below. The APL form is shown in the top left corner but obviously it would normally be hidden, preventing the user from knowing that he was using APL at all.



The Future

As you can see from Microsoft's reluctance to document the DDE hooks in their applications, DDE is on the way out. It has been replaced by OLE, *Object Linking and Embedding,* which provides much greater capabilities, but which has some strange ideas underlying it. The *embedding* aspect of OLE is about a document from one application being embedded within a document from another. It is not clear to me what relevance this has to the majority of APL systems.

Central to Microsoft's thinking seems to be that users work primarily with documents, and this has influenced the interface to a great extent. It ignores the task-orientated nature of a great deal of client/server computing. How the Windows APL interpreters implement OLE will determine whether we are funnelled down this route or can use OLE to link applications in innovative ways undreamed of by Microsoft.

The Ball Clock Problem

by Roger K.W. Hui

This article discusses Problem #1 in the Finals of the 1995 ACM International Collegiate Programming Contest sponsored by Microsoft. The problem statement is verbatim from the WWW page http://www.acm.org/~contest/clock.html; earlier versions of the text appeared in the Internet news group comp.lang.apl (ACM [1995], Hui [1995], and Weigang [1995]).

The Problem

Tempus et mobilius	*Tempus est mensura motus rerum mobilium.*
Time and motion	Time is the measure of movement.
	– Auctoritates Aristotelis

... and movement has long been used to measure time. For example, the ball clock is a simple device which keeps track of the passing minutes by moving ball-bearings. Each minute, a rotating arm removes a ball bearing from the queue at the bottom, raises it to the top of the clock and deposits it on a track leading to indicators displaying minutes, five-minutes and hours. These indicators display the time between 1:00 and 12:59, but without "a.m." or "p.m." indicators. Thus 2 balls in the minute indicator, 6 balls in the five-minute indicator and 5 balls in the hour indicator displays the time 5:32.

Unfortunately, most commercially available ball clocks do not incorporate a date indication, although this would be simple to do with the addition of further carry and indicator tracks. However, all is not lost! As the balls migrate through the mechanism of the clock, they change their relative ordering in a predictable way. Careful study of these orderings will therefore yield the time elapsed since the clock had some specific ordering. The length of time which can be measured is limited because the orderings of the balls eventually begin to repeat. Your program must compute the time before repetition, which varies according to the total number of balls present.

Operation of the Ball Clock

Every minute, the least recently used ball is removed from the queue of balls at the bottom of the clock, elevated, then deposited on the minute indicator track, which is able to hold four balls. When a fifth ball rolls on to the minute indicator track, its weight causes the track to tilt. The four balls already on the track run back down to join the queue of balls waiting at the bottom in reverse order of their original addition to the minutes track. The fifth ball, which caused the tilt, rolls on down to the five-minute indicator track. This track holds eleven balls. The twelfth ball carried over from the minutes causes the five-minute track to tilt, returning the eleven balls to the queue, again in reverse order of their addition. The twelfth ball rolls down to the hour indicator. The hour indicator also holds eleven balls, but has one extra fixed ball which is always present so that counting the balls in the hour indicator will yield an hour in the range one to twelve. The twelfth ball carried over from the five-minute indicator causes the hour indicator to tilt, returning the eleven free balls to the queue, in reverse order, before the twelfth ball itself also returns to the queue.

Input

The input defines a succession of ball clocks. Each clock operates as described above. The clocks differ only in the number of balls present in the queue at one o'clock when all the clocks start. This number is given for each clock, one per line and does not include the fixed ball on the hours indicator. Valid numbers are in the range 27 to 127. A zero signifies the end of input.

Output

For each clock described in the input, your program should report the number of balls given in the input and the number of days (24-hour periods) which elapse before the clock returns to its initial ordering.

Sample Input

30
45
0

Output for the Sample Input

30 balls cycle after 15 days.
45 balls cycle after 378 days.

A Solution In J

The balls are assumed to be labelled with the integers `i.n`. The clock period, the number of elapsed days before the clock repeats, can be computed as follows:

```
m    =. >@(0&{)
v    =. >@(1&{)
h    =. >@(2&{)
qu   =. >@(3&{)
z    =. i.@0:
ret  =. |.@}:
init =. z;z;z;i.

f1m  =. (m,{.@qu);v;h;}.@qu
f5m  =. (z;(v,{:@m);h;qu,ret@m) @ (f1m^:5)
f1h  =. (z;z;(h,{:@v);(qu,ret@v)) @ (f5m^:12)
f12h =. (z;z;z;qu,ret@h,{:@h) @ (f1h^:12)

perm =. qu @ f12h @ init
ord  =. *./ @ (#&>"_) @ C.
days =. -: @ ord @ perm
```

The basic data structure is a 4-element list of boxes (`m;v;h;qu`) of the balls in the minute, 5-minute, and hour tracks and in the queue. (The fixed ball in the hour track is ignored.) Verb `init` initializes the clock: all tracks are empty and all balls are in the queue. Verb `f1m` models the clock action every minute; `f5m` models the clock action every 5 minutes (including the action every minute); `f1h` models the clock action every hour; and `f12h` models the clock action every 12 hours.

At the end of 12 hours, all tracks are empty and all balls are in the queue; therefore, the action of the clock is a permutation of the balls, computed by the verb `perm`. The order of this permutation is the LCM of the cycle lengths of the permutation, representing the number of 12-hour periods to return to the identity, and the clock period is half this number.

The following examples illustrate the internal workings of the algorithm:

```
   days 45
378
```

Initial state for 45 balls (`m;v;h;qu`)

```
   [ s=.init 45
|||0 1 2 3 4 5 6 7 8 ... 36 37 38 39 40 41 42 43 44|
```

After 1 minute

```
   f1m s
|0|||1 2 3 4 5 6 7 8 9 ... 36 37 38 39 40 41 42 43 44|
```

After 5 minutes

```
   f5m s
| |4| |5 6 7 8 9 10 11 12 ... 39 40 41 42 43 44 3 2 1 0|
```

After 1 hour

```
   f1h s
| | |16|15 23 22 21 20 28 27 26 25 33 32 31 30 38 37 ...|
```

After 30 minutes

```
   f5m^:6 s
| |4 9 14 19 24 29| |30 31 32 33 34 35 36 37 38 39 ...|
```

After 5 hours and 32 minutes

```
   f1m^:2 f5m^:6 f1h^:5 s
|21 2|8 24 4 5 43 11|16 36 22 1 23|32 41 33 17 26 ...|
```

The queue after 12 hours for 45 balls

```
   [ p =. perm 45
25 30 0 24 35 2 44 33 15 19 13 6 29 43 8 26 40 31 ...
```

Display in 5 columns

```
   _5[\p
25 30  0 24 35
 2 44 33 15 19
13  6 29 43  8
26 40 31 12  7
38 28  3 42 32
41 14  5 37 39
 4 21 20 11 17
34 18 27 10 23
 1 22 36 16  9
```

Cycles of this permutation

```
   C. p
|26 14 8 15|42 36 18 ...|43 16 40 ...|44 9 19 7 33 11 6|
```

Column display of cycles

```
   ,.C. p
|26 14 8 15                                               |
|42 36 18 12 29 39 23                                     |
|43 16 40 1 30 4 35 34 17 31 21 28 37 27 5 2 0 25 41 ...  |
|44 9 19 7 33 11 6                                        |
```

Cycle lengths

```
   #&>C. p
4 7 27 7
```

LCM of cycle lengths

```
   *./4 7 27 7
756
```

```
   days 45                              # days is half the # of 12-hour periods
378

   d=.days"0[27+i.101                   Clock periods for 27 to 127 balls

   ,"2 [ _5 [\ (2 1$' '),' ',.-":,.d

                      23       76      102
     15       85      65      138       91
     12      117     120      345       35
     37      217     114      110      105
    378      126    6270       12      513
   1116      770      86       51       30
    693      180     930      559      858
    495    11067     714      455      378
     84      105      12      236     7095
    255       60    4524     3848     1530
   1955      268     714    25389     1155
   9360     2006     805     4446     1122
    540       36     570     1026    11033
   1218     6580     312      301     3367
  42780     2550     550     1365     6630
    105      861    4514      291     3960
   1464     1577    4284     5187      126
  17094    15330    1785    43890    25872
   5762     3325     204     3420    78120
   1224      776     273   108855      174
  14430    80080    2415
```

Permutation Power and Log

Given the current reading (m;v;h;qu) of the clock, can one determine the elapsed time since the initial operation of the clock? (Assuming that the clock has not yet repeated.)

If p is a permutation and k is an integer, the phrase q=.p&{^:k i.#p applies p to the identity permutation k times, obtaining q. By analogy with ordinary multiplication, q is the k-th power of p and (ord p)|k is the log of q relative to p. Determining the elapsed time reduces to finding k=.p log q, where p is the generator permutation of the clock (the state of the queue after 12 hours) and q is the current permutation (the state of the queue at the next even 12-hour). We proceed as follows:

First, compute the residue of q in each of the cycles of p. For example, 1 2 3 4 5 0 consists of a single cycle and 2 3 4 5 0 1 is at 2 relative to that cycle. In general, the residue of q relative to a cycle c is _1+m-((>c){q)i.{:>c [m=.#>c; moreover, if q is a power of p, the result of q indexed by the cycle, (>c){q, must be a rotation of the cycle.

For example:

```
   p=.2 3 4 5 6 7 8 1 9 0
   q=.6 3 8 5 9 7 0 1 2 4

   C. p
┌───────┬───────────┐
│7 1 3 5│9 0 2 4 6 8│
└───────┴───────────┘

   [ c0=.0{C. p
┌───────┐
│7 1 3 5│
└───────┘

   (>c0){q
1 3 5 7
   {:>c0
5
   1 3 5 7 i. 5
2
   [ m0=.#>c0
4
   _1+m0-((>c0){q)i.{:>c0
1
```

```
   [ c1=.1{C. p
┌───────────┐
│9 0 2 4 6 8│
└───────────┘

   (>c1){q
4 6 8 9 0 2
   {:>c1
8
   4 6 8 9 0 2 i. 8
2
   [ m1=.#>c1
6
   _1+m1-((>c1){q)i.{:>c1
3
```

The preceding computations can be encapsulated as follows:

```
assert=. 13!:8@(12"_)^:-.
res1   =. <:@#@[ - { i. {:@[
chkr   =. [: assert { -: res1 |. [
res    =. res1 [ chkr
mr     =. #&>@[ ,. (res&> <)
```

```
   (>c0) res q
1
```
Residue in cycle 0

```
   (>c1) res q
3
```
Residue in cycle 1

```
   (C. p) mr q
4 1
6 3
```
Moduli and residues

```
   assert 1
1
```
Return 1 if the argument is 1

```
   assert 0
|assertion failure
|       assert 0
```
Signal error if the argument is 0

```
   (C. p) mr i.-10
|assertion failure
|  (C.p)    mr i.-10
```
Not a power of p

The verb `mr` produces a 2-column table of moduli (cycle lengths) and residues. `p log q` obtains from this table by application of the GCD algorithm discovered by Euclid in 300 B.C. and the Chinese Remainder Theorem by Sun Tsu in 350 A.D. (Graham, Knuth, and Patashnik [1988]; Iverson [1995]). The Euclidean algorithm produces `a` and `b` such that `(m+.n)=(a*m)+b*n`, and the Chinese Remainder Theorem specifies that `d=.(m*.n)|(m+.n)%-(r*b*n)+s*a*m` satisfies `r=m|d` and `s=n|d`, for moduli `m` and `n` and residues `r` and `s` obtained from a power of `p`. If `cr` is a verb such that `(m,r)cr(n,s)` produces `(m*.n),d`, then the answer that we seek is `k=.{:cr/t`.

As indicated, the GCD is computed as a linear combination of the arguments; the algorithm operates by repeated remaindering. Thus:

```
g0  =. , ,. =@i.@2:
it  =. {: ,: {. - {: * <.@%&{./
gcd =. (}.@{.) @ (it^:(*@{.@{:)^:_) @ g0

   32 gcd 44                          GCD as coefficients
_4 3

   +/ _4 3 * 32 44                    The actual GCD
4

   [ a=.32 g0 44                      Initial argument for GCD
32 1 0
44 0 1

   it a                               One iteration
44 0 1
32 1 0

   it it a                            Two iterations
32  1 0
12 _1 1

   <"2 it^:(i.6) a                    All iterations; stop when 0=lower
                                      left corner
┌───────┬───────┬─────────┬──────────┬──────────┬──────────┐
│32 1 0 │44 0 1 │32  1 0  │12 _1  1  │8  3 _2   │4 _4  3   │
│44 0 1 │32 1 0 │12 _1 1  │ 8  3 _2  │4 _4  3   │0 11 _8   │
└───────┴───────┴─────────┴──────────┴──────────┴──────────┘
```

The verb `it` applies to a 2-by-3 matrix: column 0 are the two current remainders; columns 1 and 2 are the corresponding coefficients in terms of the original arguments. At each step, a new remainder is computed by using row 1 as the divisor, and the iteration stops when the divisor is zero.

The version of Chinese Remainder used here rejects residues not obtainable from a power of p. Thus:

```
ab     =. |.@(gcd/ * [ % +./)@(,&{.)
cr1    =. [: |/\ *.&{. , ,&{: +/ .* ab
chkc   =. [: assert ,&{: -: ,&{. | {:@cr1
cr     =. cr1 [ chkc
```

```
   4 1 cr 6 3
12 9
```

Applies to (modulus,residue) pairs
Produces (LCM, remainder)

```
   4|9
1
```

Verify residue 0

```
   6|9
3
```

Verify residue 1

```
   c0=. <i.4
   c1=. <4+i.6
```

A 4-cycle
A disjoint 6-cycle

```
   [ p=.C. c0,c1
1 2 3 0 5 6 7 8 9 4
```

The perm. whose cycles are `c0` and `c1`

```
   [ q=.c0 C. i.10
1 2 3 0 4 5 6 7 8 9
```

One application of cycle `c0`

```
   (C. p) mr q
4 1
6 0
```

Moduli and residues

```
   4 1 cr1 6 0
12 3
```

Chinese remainder says answer is `3`, but `1-:4|3` and `0-:6|3`

```
   4 1 cr 6 0
|assertion failure
|   4 1     cr 6 0
```

Chinese remainder with built-in check

The power and log of a permutation are now defined, together with examples which illustrate their workings:

```
pow    =. i.@#@[ C.~ (#&>@C.@[|]) # C.@[
log    =. {: @ (cr/) @ (C.@[ mr ])

   p=.4 17 7 18 10 11 0 13 8 16 9 6 15 19 12 14 3 1 2 5

   p log i.20
0

   p log p
1

   p log p{p
2
```

```
   p log p{p{p
3

   p log p pow 3
3

   [ c=.C. p                        The cycles of p
```

8	15 14 12	17 1	19 5 11 6 0 4 10 9 16 3 18 2 7 13

```
   ord p                            The order of p; LCM of the cycle lengths
42

   p log /:p                        The log of the inverse is 1 less than the order
41

   [ q=. p pow 38                   A power of p
19 1 9 4 5 2 13 16 8 6 11 7 14 3 15 12 0 17 10 18

   p log q
38

   c mr q                           Moduli and residues of q in each cycle
 1  0
 3  2
 2  0
14 10

   cr/ c mr q                       Order & log by repeated Chinese Remainder
42 38

   {: cr/ c mr q                    Select last item
38

   k -: p log"1 p pow"1 0 k=.i.ord p          Verify all powers
1

   p log ?~#p                       A random perm. is unlikely to be a power
|assertion failure                  (probability (ord p)%!#p)
|  p     log?~#p
```

The verb `pow` exploits the dyad `x C. y`, which permutes `y` by the cycles `x`. Although the definition is less direct than `p&{^:k i.#p`, the time required is exponentially less. Thus:

```
   9!:1 [7^5                        Set random seed to 16807

   p=.?~300                         A random permutation of length 300
   c=.C. p                          The cycles of p

   #&> c                            Cycle lengths
 1 7 8 1 76 89 121 6
```

```
   ord p                              Order (LCM of cycle lengths)
3862320

   [ k=.?ord p                        A random power
2908096

   (#&>c)|k                           The residues of k modulo the cycles
0 2 0 0 32 16 103 4

   (p pow k) -: (i.#p) C.- 0 2 0 0 32 16 103 4#c
1                                     Apply each cycle the residue # of times

   (/:p) -: p pow _1                  Works on all integer powers
1

   (i.#p) -: p pow 0
1

   (p pow j) -: p pow n+j=.?n=.ord p
1

   pow1=.{^:(]`(i.@#@[))              Alternative definition p&{^:k i.#p

   timer=. 6!:2                       J 2.06 under Windows on 80486/50
   timer 'q0=.p pow k'
0.05

   timer 'q1=.p pow1 k'
938.35

   q0 -: q1
1
```

That is, 0.05 seconds versus approximately 15.5 minutes. Finally, to give a sense of the relative times required for `pow` and `log`:

```
   timer 'j=.p log q0'
0.55

   j=k
1
```

References

ACM, *1995 ACM International Collegiate Programming Contest Finals, Problem 1*, http://www.acm.org/~contest/clock.html, 1995.

Graham, Ronald L., Donald E. Knuth, and Oren Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, Addison-Wesley, 1988 5.

Hui, Roger K.W., *Challenge: ACM Ball Clock Problem*, Internet News Group comp.lang.apl, 1995 3 28-31.

Iverson, Kenneth E., *Concrete Math Companion*, Iverson Software Inc., 1995 6.

Weigang, James, *Challenge: ACM Ball Clock Problem*, Internet News Group comp.lang.apl, 1995 3 27 and 1995 4 2-3.

Appendix: Collected Definitions

Clock Period

```
m      =. >@(0&{)
v      =. >@(1&{)
h      =. >@(2&{)
qu     =. >@(3&{)
z      =. i.@0:
ret    =. |.@}:
init   =. z;z;z;i.

f1m    =. (m,{.@qu);v;h;}.@qu
f5m    =. (z;(v,{:@m);h;qu,ret@m) @ (f1m^:5)
f1h    =. (z;z;(h,{:@v);(qu,ret@v)) @ (f5m^:12)
f12h   =. (z;z;z;qu,ret@h,{:@h) @ (f1h^:12)

perm   =. qu @ f12h @ init
ord    =. *./ @ (#&>"_) @ C.
days   =. -: @ ord @ perm
```

Moduli (Cycle Lengths) and Residues

```
assert=. 13!:8@(12"_)^:-.
res1   =. <:@#@[ - { i. {:@[
chkr   =. [: assert { -: res1 |. [
res    =. res1 [ chkr
mr     =. #&>@[ ,. (res&> <)
```

GCD as a Linear Combination

```
g0     =. , ,. =@i.@2:
it     =. {: ,: {. - {: * <.@%&{./
gcd    =. (}.@{.)@(it^:(*@{.@{:)^:_)@g0
```

Chinese Remainder

```
ab     =. |.@(gcd/ * [ % +./)@(,&{.)
cr1    =. [: |/\ *.&{. , ,&{: +/ .* ab
chkc   =. [: assert ,&{: -: ,&{. | {:@cr1
cr     =. cr1 [ chkc
```

Permutation Power and Log

```
pow    =. i.@#@[ C.~ (#&>@C.@[|]) # C.@[
log    =. {: @ (cr/) @ (C.@[ mr ])
```

Is APL a Team Sport?

by Douglas R. Bohrer, Kemper Securities Inc.

Abstract

The article discusses ways in which APL programming teams can be managed to complete large, time-sensitive projects using parallel development. All aspects of the team effort are considered including design considerations, tool-kit use and staffing and organisation. The concepts described were used with success during the author's five-year consulting engagement with a large retailer.

Do We Really Need a Team?

Many APLers view APL as so powerful that only one or two of us are needed to do any project, no matter how large or complex. However, I think that a combination of complexity, lack of project definition and extreme time pressure can make a project impossible without a team. Let me give you some background on the business situation so you can understand my experience with APL teams.

The projects I had to do were all merchandise planning systems with a four to six month development window. They all started life with a one or two page management directive requiring a new system or major enhancement in very general terms. The only clear requirement was the deadline. The main objective was to make it as painless as possible for company management in hundreds of stores, with over forty categories of merchandise, to plan sales, inventory, profits or losses.

When I started, I was hired to get a creaky ADRS (A Departmental Reporting System) based annual planning system through one last planning cycle while all of the previous programmers were being transferred to other areas. The PL/1 development of a replacement system was running late. I got through it with a lot of help from the folks waiting to be transferred and a lot of overtime. The job was just too big for me alone, or even for two people. Possibly Superman could have done it without help, but he wasn't available. We had to have a team.

The second effort was a major redevelopment with a six-person team. The PL/1 development was going to be even later, so the current system had to be used yet again one more time. Our success in this effort led management to cancel the PL/1 project. Instead, we would develop the added functionality in the old system. Once that was completed, we were given a lot more assignments. The APL team got as big as 21 people doing multiple projects. Personally, I ended up with a five-year extension of a two-month contract. As we did more and more, I saw a pattern to our successes, which I describe below.

Design Considerations

The first step to each of our projects was to get the project defined. "What's it gotta do?" is the first question to be answered. Once the desired behaviour was described, the APL team had to answer "How ya gonna make it work?"

I found that the best way to get the users to describe what they wanted was to work first on what they wanted to see. We could mock up menus and menu paths first. Then we would design the interactive screens and the calculations that supported them. The next step was the reports and the calculations needed for them. Once we had what the user wanted to see, we could then go through each screen and report to ask where the data would come from. The data sources usually would lead us to reading files from other systems. Finally, we would ask what data had to be sent to other systems.

Once there was a fairly complete definition of what the system should do, we had to talk about deadlines. Using the menu paths, we would define which options needed to be available immediately and which could wait. For example, reading last year's actual sales had to be ready the day the system was installed, but exporting the plan data to accounting systems could wait. Because of the time pressure, development effort had to focus on the immediate needs, with development of delayed features while the users were using the initial product.

Specification Changes

Planning systems seem to suffer from a lack of planning, possibly for the same reason that a shoemaker's children lack shoes. Specification changes during development were frequent. I found that the best way to handle them was first to ask often about future directions, and second to discourage changes enough to make sure proposed changes were worth fighting for.

The users were reluctant to reveal what new features might be required until the planning methodology was approved at the highest levels. However, over time I was able to get them to reveal 'hypothetical' directions so that I could allow for them in the design. In exchange, I would provide 'ball park' estimates on how hard it would be to make the 'hypothetical' changes. The team also kept their mouths shut about the information.

Discouraging last minute changes requires a light touch. I always mentioned our motto, *"We're in a hurry!"*. I did this so often that the users would quote it to me before they asked for a change. I would then tell them how much the change would cost, particularly in terms of time. The easy stuff I would do quickly. The hard stuff I would try to negotiate into something functionally adequate that was easier to program. In particular, I would try to keep changes within the capabilities of the existing tool-kit.

Technical Design

The primary design consideration is to maximise parallel development. This means that dependencies between functions have to be minimised by design. The key design concept to minimise dependencies is the *hub* for data flows. Just as airlines use a hub airport to move their passengers, the APL team should use pre-defined global workspace variables as a hub for all data flow. Restrict **all** data movement to movement into or out of the hub variables. In summary, to go to hell you have to change planes in Atlanta.

I mean **all** data movement goes through the hub variables: file I/O, screen entry and display, reports and even data imports and exports. Make as few exceptions as possible. In effect, the hub variables and the utility functions that get and set the hub allow object-oriented programming.

Let me illustrate by describing the data flow in a typical interactive screen. The screen driver program uses file I/O functions to fill the hub variables. It calls common calculation functions which use and set hub variables. Toolkit functions get data from the hub variables for special calculations using local variables. The screen driver then puts the results back into the hub using tool-kit functions. A screen-handling utility displays hub contents, and updates the hub from screen entries.

Benefits of Hub Concept

Using a hub maximises parallel development by making each piece of a project independent of every other piece. Each feature depends only on the hub, and is specified in terms of its interaction with the hub. Screen development, for example, can begin before there are any files because the screen driver program interacts with the hub. In the actual projects, screen development often started before there were any test files.

Using a hub also improves reliability. Testing can be done on each piece of code because each is independent. Results are verified in observed effects on the hub variables. Once verified, more code is re-usable because it works on the hub in the same way whenever it's used. For example, a gross profit calculation using and setting the hub can be called from both screens and reports because they all use the hub.

Tool Kit Construction

My basic philosophy on tool-kit functions is that they have to be easy-to-use black boxes. It has to be easy to learn how to use them or they won't get used. Over half of the people I worked with were short-term consultants hired specifically for the four to six month project window. They needed to become productive as fast as possible, so the tools had to be easy to learn. (Remember our motto.)

I started with the functions provided by ADRS. The hub started as the global variable structure of ADRS and the functions ADRS used for the structure. I gradually replaced the file system, the screen handlers and the reporting system by liberally borrowing from whatever was lying around. Improvements were based on customer requests for new features or persistent performance bottlenecks. While the specifics are probably obsolete, an example is worth discussing to show how the evolution was eased by having the hub to work with.

File I/O in ADRS, with VSAPL under MVS/XA, was slow because there was far more data than ADRS was ever designed for. In addition, allowing for multiple users had been accomplished by splitting the data into multiple small files which users could lock exclusively for update. All the files were read in a loop for combined reports. Starting with the functions in library 2 VAPLFILE using AP121, I replaced the multiple ADRS files with component pseudo-files in VSAM clusters. File locking was based on exclusive control of QSAM "shadow" files, one "shadow" for each component pseudo-file.

At the highest level, the new file functions set and used the hub variables, just like the ones they replaced. The only differences were in behaviour outside the hub. In the old file structure, multiple small files were segregated using hub values. In the new file structure, pieces of the component files were locked by using the same hub values stored on the file. None of the application code other than the file function calls changed at all.

The new arrangement changed clock time for reports from 40 minutes to three minutes. The file functions were designed to make a switch to real component files relatively painless. When we finally did switch to a commercial component file server, no code above the file functions changed at all. The hub variables were set just as before. True component files made the file-read time for the big reports about 15 to 30 seconds.

The advantages of using a tool-kit are many. Reliability is enhanced because the code is debugged thoroughly then re-used a lot. Flexibility in addressing performance problems comes from modular replacement, like the file I/O evolution described above. Application maintenance is also easier because everybody knows what the tools do and can follow the code by following the tool use. Coding speed is increased because you only invent the wheel once. Training is simplified because initially people only have to learn what the tools do and can leave how they do it for later.

Staffing and Organisation

Once you have more than a few people, staffing and organisation become almost as important as purely technical considerations. You have to interview, train and structure a team of notoriously quirky APLers. You have to maintain discipline and morale. You also have to handle a lot more external politics because a team is much more visible than the usual 2–3 APL people in the back closet.

The first step is interviewing new prospects. The most important thing here is to be yourself. You want the prospect to get a good idea of what working with you will be like so he can decide whether to take the job or not. Start by explaining the job. It's a good idea to have several people gang up on the applicant so that the applicant gets a more complete picture of who he'll work with and what he'll be doing. It also gives several people in your shop an idea of whether or not they want to work with the applicant. I always gave a little test, asking for an explanation of a few lines of APL. This gave me a surprisingly accurate view of how well the prospect knew APL. However, I found that a real burning desire to work in an APL shop was probably more important than specific knowledge. I found that good training expanded our choices.

I did most of the training personally. I did this to show the newcomer that training is important. My involvement allowed me to evaluate potential assignments for the trainees, but more importantly, allowed the trainee to get used to working with me. Remember that training is both social and technical. On their first day, everybody is a trainee even if they have lots of technical experience. They don't know your application and they don't know how to work with you.

At the start of training, I had everybody work through *APL: An Introduction* by Howard Peelle. This served as a review, since many of them had not been working in APL recently. While the book may be out of print, any similar introduction would do. The idea is to get everybody up to speed in APL without embarrassing anybody. Working at their own pace, trainees took from 2 days to 2 weeks to get through the book and some selected problems. I never made a big deal about how long it took. I did mention often that once through the book, everyone was starting more or less even in terms of his exposure.

It is important to give realistic training problems so that the trainee knows you are not wasting his time. The realism should include incomplete specification of the problems. You can do this in a fairly humorous fashion. I always used to tell trainees that I was going to simulate our typical users in specifying problems, so they had to ask me for clarification if they needed it. Then the first time I caught them assuming something I hadn't told them, I would get very wide-eyed and ask, "Whatever gave you that idea?". I did this even if the assumption was OK, just so they would learn to ask.

One good training exercise is to have trainees write new tool-kit functions you always wanted or re-write functions that are too slow. If the results are good enough, and they usually are, you can install the new code. For example, I needed a menu generator which would take a character matrix and put it on the screen, making every underscore character an entry field. My boss thought this would be too difficult to be practical. (We were using GDDM.) I had two trainees do it to my specification, then installed it. It was a big boost for their morale to see their "graduation exercise" used by everybody on the team. It also demonstrated feasibility without taking any experienced people off time-critical projects.

Team Structure

When you are structuring an APL team, you have to remember that 1 man-month of APL code is a LOT of functionality. APL teams tend to have more design and management work per coder than programming teams using other, less productive, languages. I found that I needed a foreman for every 3-5 people. Foremen came in two varieties, an architect/designer who didn't code much and a general contractor who coded about half-time. The architect handled most of the user-contact during development and designed like mad trying to keep ahead of the coding. The general contractor answered most programmers' detailed questions during coding, contacting the architect or the users on points that had yet to be decided.

I found that people tended to develop functional, rather than project, specialisation. Some were really good at reports, some were super screen specialists and some were great at getting data from other systems. People would switch from project to project often, but would usually do the same area on each project.

Functional specialisation seemed to be popular with the team. It also allowed rapid redeployment from one project to the next, which was usually a scheduling requirement. Team members understood their functional area completely and found it easy to apply their knowledge to new projects. The methods in each project tended to converge toward a *de facto* standard. As the same people worked on similar problem areas across multiple projects, they found it easier to use familiar tools in new situations.

Discipline and Morale

APLers are special characters... a sense of humour is required. For example, if everybody is writing his own tools instead of looking what's available first, you can't just order everyone to look before writing. You instead suggest that the Committee of Public Safety and Programming Practice, fresh from its efforts in the recent (1789) French Revolution, is now in control of approving new tool-kit functions. The penalty for submitting a duplicate function could be a pain, given their history. Well, at least it worked for me.

It's best to explain why something has to be done, and not just what has to be done. Then you can ask for advice, which you should do whenever you have time for it. You have to be approachable, particularly if you're the architect/ designer. The specification is not going to be complete unless questions are asked.

Everybody has to be approachable on past mistakes. If you have a problem with others finding your mistakes, remember our motto. We were in a hurry when you messed up.

Some things will go right, some wrong. When they go right, broadcast compliments loudly. When they go wrong, let the punishment fit the crime. Remember, you want solved problems not fear and trembling. For example, when I got beeped at 3 a.m. about a bug in the overnight job, I didn't yell at the programmer. I just walked into his office the next morning and put the pager on his desk. I quietly told him the beeper was his to carry until he got a good night's sleep with no interruptions. Two nights later the code worked perfectly, and he gave me the beeper back.

Politics

Most programmers I've met fear and loathe politics, probably because they don't really understand how politics works. For some reason, I've always enjoyed politics and have managed to do fairly well at playing them. The key thing to remember in the politics of programming is that programming exists to provide valuable information with the least possible problems. To find out what's valuable, you have to talk to management and your users. You also have to minimise their problems in dealing with you and your system.

APL team political problems come up dealing with three types of people outside the team. The most immediate, if not most important, is the team's non-APL boss, who can barely spell APL before he gets stuck with a very strange collection of subordinates. The second group includes all the poor desperate users of your systems. They are stuck with a new or vastly altered system with advance billing that reminds them of the last three unnatural systems disasters they lived through. The last type, the most important type of all, is the client / systems administrator who was selected by senior management to explain their grand idea to the ignorant slobs in the systems department. Each type has special needs, which I will describe based on my results of trying a lot of things that didn't work before stumbling on something that did.

Your Non-APL Boss

This individual probably has a background in data processing management and is used to a normal pace of development. The speed with which you do things will pleasantly surprise him. How you do it will make him uncomfortable. Always tell him what you are doing, describing the features as the users will view them. Avoid telling him how these things are coded in APL. Use as little jargon as possible so you can be sure he understands what you tell him and never gets the impression you are trying to baffle him with fancy-sounding baloney. Especially avoid APL jargon, since he is unlikely ever to have heard it and will begin to wonder if you speak English (or the local equivalent).

While there are many things about APL that might require special treatment from the systems bureaucracy, never ask to be a special case unless there is NO WAY you can avoid it. Remember that special cases are special trouble for your manager. Save your special requests for things you will die without.

Your Poor Desperate Users

These people may never have seen a system that they liked using. They are used to rude responses from data processing every time they find a bug in the code and may have given up on reporting them. They will love you if you admit your mistakes immediately. You should encourage error reporting. Offer to fix the first guy's data for free. You have to test anyway, so you might as well use his data. Follow up after you put out the fire and ask for suggestions. Your quick response will contrast starkly with what users have seen before and you will see a big change in their attitude.

Using this approach I quickly became very popular. Users became almost deferential, saying things like, "I know you're really busy, but I think I may have found something that doesn't work right." Secretaries would say things like, "I KNOW he'll talk to you, please wait while I find him." I found out that many of the managers I talked to routinely were notorious for not taking phone calls from mere peons, like my boss's boss.

Your Client / Systems Administrator

[Author's note to the Pronoun Gender Police: female pronouns are used in this section because the majority of the people I dealt with in this position were women. If you are sensitive about pronouns which reflect reality, I apologise.]

You are wondering how she got this job? She was drafted! She probably had a succession of increasingly important jobs with managerial responsibility in the user area. Now her success depends on you, and you look weird to her. She doesn't know what to expect and the lack of control scares her. The trick here is to be reassuring without over-promising. Never make anything look or sound easy. Tell her doing the job on time will be tricky, a LOT of hard work, but you have been successful before and expect to be this time as well.

It is better to be early than late, so be careful when making time estimates that there is some room for unexpected problems. In scheduling, remember the specifications are going to change and make allowances. She has to publish your completion schedule and explain to her management when the schedule slips, so it is very important to make the schedule realistic.

If you solicit rumours of changes in requirements, you can provide her with rough guesses about how they might change the schedule. Then she can use the estimates to force her management to decide if the changes are worth the delay. No matter how they decide, everyone involved will feel more comfortable having exchanged the information.

Encourage her to be as loud about your successes as she is about your problems. Constantly remind her, your motto is "We're in a hurry!".

Multiprecision Arithmetic: Part II

by John Sullivan

The Story so Far

In Part I, I explained my motivation for writing a multiprecision arithmetic suite in APL, and I gave the code for the addition, subtraction, multiplication and division functions, together with some other goodies, such as changing the radix of a number and raising a multiprecision number to a small power. I also showed the matrix multiplication way of generating Fibonacci numbers using the inner product operator with two user-defined functions.

Now read on ...

Input and Output

If we set our radix to 10^8 and we multiply 0 11 by 0 9090910 we get 0 1 10, which is pretty meaningless. We need a function to convert our output to a suitable format so that we can read it. Similarly, we need an input function, because it is a lot easier to input large numbers in character format, with blanks or commas to group the digits in suitable blocks, than it is to input them directly as integers.

The formatting function is called `Ffmt` (Float format). It takes a scalar left argument, which indicates the size of the blocks into which the number is separated. If this is positive the separator is a blank, if negative it is a comma. For example:

```
      ¯3 Ffmt ¯1 12345 67890000
12,345.678,9
      3 Ffmt ¯1 12345 67890000
12 345.678 9
      0 Ffmt ¯1 12345 67890000
12345.6789
```

Since we are preparing human-readable output, the radix of our operations should be a power of 10. For the purposes of this function I am going to change the radix to 10^8 if the radix is not already a power of 10.

```
      ∇ z←{h}Ffmt a;b;x;k;n;p;s
[1]      →(0=1|p←10⍟base)/Δ1
[2]      a←(100000000,base)chbase a ⋄ p←8
[3]     Δ1:p←⌊p
```

We now strip off the number of radix places, and drop trailing zeros.

```
[4]      n←a[0] ⋄ a←1↓a
[5]      a←(-s←(0≠⌽a)⍳1)↓a ⋄ n+←s
```

But, if we have an integer with a positive exponent, we put the trailing zeros back, or add new zeros if there weren't any.

```
[6]      →(n≤0)/Δ2
[7]      a,←n⍴0 ⋄ n←0
```

Get the sign of the number. If it is zero, then make a quick dash for the exit.

```
[8]     Δ2:→(0≠s←×1↑a←((a≠0)⍳1)↓a)/Δ3
[9]      z←'0' ⋄ →0
```

Having got the sign we now work with the absolute value of the number. If the number is fractional and we need leading zeros then supply them.

```
[10]    Δ3:a←|a
[11]     →(n≥0)/Δ4
[12]     a←(n⌊-⍴a)↑a
```

Format each part of the number into blank-separated blocks of numeric characters with leading zeros. Each block is p characters long, where p is $\log_{10}$ of the radix. If we have a fraction then put the decimal point in the appropriate place (this is made easy by the blanks between the blocks of numbers.

```
[13]    Δ4:z←,' ',('ZI',⍕p)⎕FMT a
[14]     →(n≥0)/Δ5
[15]     z[(⍴z)+n×p+1]←'.'
```

Tidy up. If our number is not completely fractional drop the leading blank (line 16). If the number starts with one or more zeros, drop them (line 17), and replace one zero if this leaves the decimal point in the first character (line 18). If the number is negative put a minus sign at the start (line 19). If we have a decimal point then drop any trailing zeros (lines 20 and 21).

```
[16]  ∆5:z←(' '=z[0])↓z
[17]   z←((z≠'0')⍳1)↓z
[18]   z←(('.'=z[0])/'0'),z
[19]   z←((s<1)/'-'),z
[20]   →(~'.'∊z)/∆6
[21]   z←(-('0'≠⌽z)⍳1)↓z
```

If no left argument (*h*) was supplied, or if its value would result in no change to the appearance of the number as we have it now, then we have finished. If *h* is minus the size of the radix then we can bypass the code that puts the blanks into the right places.

```
[22]  ∆6:→(0=⎕NC'h')/0
[23]   →(h=p×1 ¯1)/0,∆8
```

Get rid of all blanks. If *h* is 0 then we have finished. The base from where the separators (blanks or commas) are measured is the decimal point, so find that. Calculate a boolean vector that can be used to put blanks in the number: line 27 does it before the decimal point and line 29 after. Line 28 handles the case where we have an integer with no decimal point.

```
[24]   z~←' '
[25]   →(0=k←|h)/0
[26]   n←z⍳'.'
[27]   s←⌽(⌊n×b←(k+1)÷k)⍴c←(k⍴1),0
[28]   →(n=⍴z)/∆7
[29]   s←s,1,(⌊((⍴z)-n+1)×b)⍴c
[30]  ∆7:z←s\z
```

Drop the resulting leading and trailing blanks. If there is a blank between the minus sign and the number then remove that also.

```
[31]   z←(' '=z[0])↓z
[32]   z←(-' '=¯1↑z)↓z
[33]   →('- '∨.≠2↑z)/∆8
[34]   z←'-',2↓z
```

If *h* was negative then we want comma separators, and that's it.

```
[35]  ∆8:→(h≥0)/0
[36]   z[(z=' ')/⍳⍴z]←','
     ∇
```

The input function is called `Fexec` (Float execute). If the number is already numeric we get out quickly. The same comments about the radix for `Ffmt` apply to this function also. Be careful here with the terminology: we have two meanings of the word *exponent*: as the number that comes after the E in a number like 1.234E5, and as the number defining the position of the radix point in the multiprecision array. I have tried to avoid confusion, but I have probably failed!

```
      ∇ x←Fexec x;e;f;i;n;p;p1;p10;q;s
[1]      →(~(1↑x)∊⎕AV)/0
[2]      →(p10←0=1|p←10⍟base)/Δ1
[3]      p←8
[4]     Δ1:p1←(p+1)÷p←⌊p
```

It is sometimes convenient to enter numbers in scientific notation, i.e. with an E to indicate multiplication by some power of 10. We handle that here. It is an error if there is more than one E (lower-case is also acceptable) in the number. Split the exponent off, drop any leading plus-sign, check that the exponent is all numbers with an optional leading minus, make it numeric (this function will crash here if the exponent is just a minus sign: this is deliberate, to allow the calling function to handle the error). The value of the exponent is stored in e, which will be 0 if there was no exponent. (Note: if your APL does not have ⎕D you can replace it by '0123456789').

```
[5]      →(0=e←+/n←x∊'Ee')/Δ2
[6]      ⎕SIGNAL(1≠e)/11
[7]      q←n⍳1
[8]      e←(q+1)↓x ⋄ x←q↑x
[9]      e←('+'=1↑e)↓e
[10]     ⎕SIGNAL(~∧/(e∊⎕D)∨(⍴e)↑(1↑e)∊'¯-')/11
[11]     e←⍎e
```

Now we basically repeat on the main part of the number what we did to the exponent, but we also have to check on the decimal point if it is present (there must not be more than one of these), and we split the number into two at the decimal point. In this part of the function *s* is the sign of the number, *i* is the integer part and *f* is the fractional part.

```
[12]    Δ2:x←(x∊⎕D,'.-¯')/x
[13]     x←(s←x[0]∊'-¯')↓x
[14]     →(0=⍴x)/0
[15]     i←(q←x⍳'.')↑x
[16]     →(0=⍴f←(q+1)↓x)/Δ3
[17]     ⎕SIGNAL(f∨.='.')/11
```

Set up the result (null vector). If there is an integer part, convert it to numerics by spacing it out according to the size of the radix, catenate this to the result. Do the same with the fractional part, which may have to have added trailing zeros in order to ensure that the last block is the correct length.

```
Δ3:x←θ
 →(0=ρi)/Δ4
 x←x,⍎(ϕ(⌊p1×ρi)ρ(p+1)↑pρ1)\i
Δ4:→(0≠ρf)/Δ5
 f←θ ⋄ →Δ6
Δ5:f←,⍎((p1×ρf)ρ(p+1)↑pρ1)\f←f,(p|-ρf)ρ'0'
```

Put the number all together, multiply all but the first element (which is the location of the radix point) by the sign of the number. Drop leading and trailing zeros.

```
Δ6:→(0∧.=x←(-ρf),1 ¯1[s]×x,f)/0
 x←(-f←(0≠ϕx)⍳1)↓x
 f←x[0]+f
 x←f,((x≠0)⍳1)↓x←1↓x
```

If the radix we used in this function is not the same as the global radix then recalculate the number in the global radix.

```
 →p10/Δ7
 x←(base,100000000)chbase x
```

Process the exponent if it is non-zero. This is done the lazy way, by generating a character number of the appropriate size and recursively calling this function. (This is the reason why this function is in part 2: we needed to define *Fmul* first.)

```
Δ7:→(1+×e)⊃Δ8,0,Δ9
Δ8:x←x Fmul Fexec'0.',((¯1-e)ρ'0'),'1' ⋄ →Δ9
Δ9:x←x Fmul Fexec'1',eρ'0'
∇
```

Square Roots

There are several algorithms for extracting square roots, but they all seem to have disadvantages somewhere along the line. I had to make a choice of algorithm here, and the one I have chosen seems to be the fastest function around, but it uses multiprecision floating-point numbers and we have to be careful to prevent the intermediate results blowing up and giving *WS FULL*. IBM[3] gives an algorithm using long division that is only slightly slower than this version, but it only works with even radices (because at one point the radix is divided by 2), whereas this version works with all radices as defined in part 1. For yet another algorithm, which is slower than these two, see [2].

As with division there are two functions for square roots, one for integer square roots with remainder, and the other for floating square roots, rounded to the nearest value. Again, as with division, the floating function calls the integer function and massages the results.

Isqrt (Integer square root)

First we make sure our input is multiprecision, then we check that it is not negative. It's also an error in this function if the input is not an integer.

```
      ∇ z←Isqrt a;b;f;n;q;r
[1]     a←scalar a
[2]     ⎕SIGNAL(0∨.>a)/11
```

If our number is 0 then we can exit at once.

```
[3]     →(0∨.≠1↑a)/Δ1
[4]     z←(0 0)(0 0) ⋄ →0
```

Use the full precision of the number.

```
[5]   Δ1:a←fullint a
```

If the square is small we can use APL's built-in exponentiation primitive, rounding the result, then go to the end for processing of the remainder.

```
[6]     →(3<ρa)/Δ2
[7]     z←0,⌊(base⊥a)*0.5 ⋄ →Δ8
```

First, we get an accurate starting value. Rather than just looking at the first pair of "digits", we try to take as much as we can of the number for the starting value. For example, in base 100, if we try to extract the square root of $(10\ 73\ 74\ 18\ 24)_{100}$, our starting value is $(3\ 27\ 68)_{100}$, rather than the 3_{100} we would get from just looking at 10_{100}. If we decide to work in base 10 (heaven forbid!) the starting value is the same, i.e. $(3\ 2\ 7\ 6\ 8)_{10}$, which is still a lot better than the 3_{10} from $1_{10}0_{10}$. And in both these cases the starting value is the square root, so we can exit straight away having saved ourselves a lot of unnecessary effort. The `0 Fadd` on line 9 is the quickest way to convert the scalar starting value to a multiprecision number.

```
[8]   Δ2:n←(ρa)⌊(2|ρa)+2×1⌈⌊8÷10⍟base
[9]     z←(n↑1+⌊0.5×ρa)↑0 Fadd⌊(base⊥n↑a)*0.5
```

Have we hit upon the square root straight away?

```
[10]    →(0 0≡r←a Fsub z Fmul z)/Δ9
```

Now we have the usual Newton-Raphson iteration: $z_{n+1} = z_n + (a - f(z_n))/f'(z_n)$

```
[11]  Δ3:b←z Fadd(a Fsub z Fmul z)Fdiv z Fmul 2
```

If we have more than 2 radix places we should ignore the extra (because they are not really needed, they slow down the algorithm, and they can give rise to *WS FULL* as we iterate).

```
[12]   ⍎(b[0]<¯2)/'b←¯2,1↓(2+b[0])↓b'
```

If the integer part of this iteration equals the integer part of the previous iteration we have finished, otherwise iterate again.

```
[13]   →((floor b)Fequal floor z)/Δ4
[14]   z←b ⋄ →Δ3
```

Drop the fractional part of the result, calculate the square-root remainder, catenate it to the result, and exit.

```
[15]  Δ4:⍎(0>z[0])/'z←floor z'
[16]  Δ8:r←a Fsub z Fmul z
[17]  Δ9:z←z r
     ∇
```

The called function *floor* removes the fractional part of the number (just like ⌊):

```
     ∇ z←floor z
[1]    →(z[0]≥0)/0
[2]    z←0,1↓z[0]↓z
     ∇
```

Fsqrt (Floating-point square root)

The floating square root function starts off the same way as the integer version.

```
     ∇ z←Fsqrt a;b;d;r
[1]    a←scalar a
[2]    ⎕SIGNAL(0∧.>a)/11
[3]    →(0∨.≠1↓a)/Δ1
[4]    z←0 0 ⋄ →0
```

We now pretend that our numbers are integers, and perform the integer square root. The number of radix places must be even, so add an extra trailing zero if they are not.

```
[5]    Δ1:→(~2|d←a[0])/Δ2
[6]     d←d-1 ⋄ a←a,0
[7]    Δ2:z r←Isqrt 0,1↓a
```

If the remainder is greater than the square root add 1 to the root. Put the radix point in the correct place.

```
[8]     →(~r Fgt z)/Δ4
[9]     z←z Fadd 0 1
[10]   Δ4:z[0]←z[0]+⌊0.5×d
      ∇
```

Raising to a Power

In part 1 I showed how to raise a multiprecision number to a small power. In many algorithms we need to raise a small number to a multiprecision power, taking the result modulo some prime p (if we do not take the result modulo p we could soon get `WS FULL`).

The following function takes a scalar or multiprecision number *m*, and raises it to the power of `x[0]` modulo `x[1]`. The method is a variant of the small-power function shown in part 1, and is described in Ribenboim[1], p.38.

First we separate the right argument into power and modulus. Convert the power into a bit string. If this is null then our power was 0 and the result is 1, so we can exit.

```
      ∇ z←m Impow x;i;mod;n;q;r
[1]     x mod←x
[2]     z←1
[3]     →(0=⍴n←binary x)/0
```

If the power is 1 then the result is the number we started with. We do not reduce by the modulus since we assume that the programmer has ensured that the number starts off smaller than the modulus.

```
[4]     z←m
[5]     →(1=⍴n)/0
```

Square `z`. Starting at the left, look at the bits of the power. If the current bit is 1 we also multiply *z* by the number *m*. Then take the residue modulo *mod*. Proceed this way until the right of the power is reached.

Of course, the first bit in the power is always 1, so we can cut some processing, and we get the following:

```
[6]    i←1
[7]   Δ1:z←z Fmul z
[8]    →(~n[i])/Δ2
[9]    z←z Fmul m
[10]  Δ2:z←z Imod mod
[11]   →((⍴n)>i←i+1)/Δ1
      ∇
```

The two functions called by *Impow* are as follows:

binary converts a multiprecision integer to bits by changing its base to a power of 2, then converting each element of the resulting number to bits. Note that we do not check that the number is a multiprecision integer: the function will fail with a *DOMAIN ERROR* if it is not.

```
      ∇ z←binary x
[1]     z←(1073741824,base)chbase x
[2]     z←(z⍳1)↓z←,⍉(30⍴2)⊤z←1↓z,z[0]⍴0
      ∇
```

Imod returns the residue of one number modulo another. It is, of course, possible to extract this result by using the remainder from *Idiv*, but there is a lot of processing in that function that is wasted when you only want the remainder, so *Imod* has been coded to run much faster.

```
      ∇ a←x Imod b;b1;j;q;qf;s
[1]     x←scalar x ⋄ b←scalar b
[2]     ⎕SIGNAL(0∨.>b[0],x[0])/11
[3]     s←0 ⋄ a←x
[4]     →(2<⍴,b←fullint b)/Δ1
[5]     →(b[1]≤bsqr)/Δ7
[6]    Δ1:b1←b[1]+(3↑b)[2]÷base
[7]    Δ2:→(~0 0 Fgt a)/Δ3
[8]     a←|a ⋄ s←~s
```

If line 9 signals an error then there is a programming error in the function. This hasn't happened for me yet!

```
[9]    Δ3:⎕SIGNAL(a Fgt|x)/11
[10]    →(~(a Fgt 0 ¯1)∧b Fgt a)/Δ5
[11]    →(~s∧~a Fequal 0 0)/Δ4
[12]    a←b Fsub a
[13]   Δ4:→0
[14]   Δ5:q←⌊qf←(a[1]+(3↑a)[2]÷base)÷b1
```

```
[15]   j←0
[16]   →(0≠q)/Δ6
[17]   q←⌊qf←qf×base ⋄ j←1
[18]  Δ6:a←a Fsub q Fmul(a[0]+(⍴a)-j)↑b
[19]   →Δ2
[20]  Δ7:b←b[1] ⋄ x←fullint x
[21]   a←x[1] ⋄ x←2↓x
[22]  Δ8:a←b|a
[23]   →(0=⍴x)/Δ9
[24]   a←base⊥a,1↑x
[25]   x←1↓x
[26]   →Δ8
[27]  Δ9:→(~s∧0≠a)/Δ10
[28]   a←b-a
[29]  Δ10:a←0,a
     ∇
```

A User-defined Operator

I suggested last time that the Fibonacci calculation could go much quicker if we wrote our own scan operator. This is because the primitive scan operator does far too much work for our purposes. As an example, consider the following function:

```
     ∇ z←a plus b
[1]    i←i+1
[2]    z←a+b
     ∇
```

If we perform `plus` reduction on the first 10 numbers, we get the following:

```
       i←0
       a←plus/⍳10
       i
9
```

But scan in the same position is horrifying:

```
       i←0
       a←plus\⍳10
       i
45
```

And this is what is happening in our Fibonacci example. Normally we do not mind the extra processing all that much, because it does not take too much time, but with the multiprecision suite it takes up far too much.

For these circumstances we need a new operator, like this one:

```
      ∇ z←(F scan)x;y
[1]     z←⊂y←⊃x
[2]    Δ1:→(0=⍴x←1↓x)/0
[3]     z,←⊂y←y F⊃x
[4]     →Δ1
      ∇
```

This operator only works on vectors, and `F scan 1 2 3 4` is equivalent to

```
      1 (1 F 2) ((1 F 2)F 3) (((1 F 2)F 3)F 4)
```

In particular it does not replace \ (try `-\1 2 3 4` and `-scan 1 2 3 4`). However, it will speed up the operation we are doing here considerably.

```
      Fadd.Fmul scan 10⍴⊂2 2⍴1 1 1 2
```

And there's more ...

You now have all of the "basic" functions: the "building blocks" of the multi-precision suite. If you have Dyalog APL you could have keyed these into a namespace called *mp* in order to keep them out of the way; if your APL does not support namespaces the functions will just be available in the workspace.

In Part III I shall introduce some applications of these functions.

References

[1] Paulo Ribenboim, *The Book of Prime Number Records* (Springer, 1988)

[2] George McCarty, *Calculator Calculus* (EduCALC Publications, 1975)

[3] *The APL Handbook of Techniques* (IBM DP Scientific Marketing, 1988. Order No. S320-5996-0)

THE RANDOM VECTOR

Computing Clopper-Pearson Confidence Limits by the Illinois Method

by Dietrich Trenkler (University of Osnabrück)
Email: trenkler@rols1.oec.uni-osnabrueck.de

Introduction

The method of Clopper/Pearson [2] to construct confidence bounds for a probability p is covered in several text books, see e.g. Bickel/Doksum [1] or Mood/Graybill/Boes [4]. Explicit formulas exist only for marginal cases and their graphical determination from charts as the ones published in Pearson/Hartley [5] is very cumbersome and imprecise. Furthermore, well-known approximations based on the normal distribution may lead to unreliable results especially when the sample size is small.

In this note we present programs written in APL to compute these intervals. The first one, called *ILLINOIS*, has proven to be efficient in those cases where the determination of a solution of an equation $f(x) = 0$ is desired without the use of its derivative f'. As the construction of the Clopper-Pearson intervals amounts to solving such an equation and the derivative of the function involved is complicated we resort to this method.

The Method and the Program

Let us start with a description of our goals. A Bernoulli trial is an experiment which can end in exactly two outcomes F (failure) or S (success). It is assumed that these Bernoulli trials are independently performed n times (the sample size) such that the number X of successes follows a Binomial distribution with probability function

$$P(X=x) = \binom{n}{x} p^x (1-p)^{n-x},\ x = 0,1,2,\ldots,n$$

where $p = P(S)$ is the probabilty of S. It is intended to use the observation $(X = x)$ to construct a confidence interval $[\hat{p}_l, \hat{p}_u] = [\hat{p}_l(X), \hat{p}_u(X)]$ for p such that

$$P(\hat{p}_l \leq p \leq \hat{p}_u) \geq 1-\alpha \tag{1}$$

where $1-\alpha$ is a confidence level chosen in advance, usually 0.95 or 0.99. Note that $\hat{p}_l(X)$ and $\hat{p}_u(X)$ have a discrete distribution as they depend on X. This means that one can compute $n+1$ pairs of confidence limits for a given sample size n.

If n is large and p not too small we can use a normal approximation and compute well-known confidence intervals from

$$I = \hat{p} \pm z\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

with $\hat{p} = X/n$ and z is the $\alpha/2$ quantile of the standard normal distribution. This interval is not exact as it can happen that $P(p \in I) < 1-\alpha$ for certain values of p which is especially true when n is small or $p \approx 0$ or $p \approx 1$. On the other hand, it can be shown that inequality (1) holds whenever $\hat{p}_l$ is chosen as the solution of

$$G(p\,|\,n,x-1) = 1-\alpha/2,\ 0 < x \leq n \tag{2}$$

where

$$G(p\,|\,n,x) = P(X \leq x) = \sum_{i=0}^{x} \binom{n}{i} p^i (1-p)^{n-i}. \tag{3}$$

Analogously, $\hat{p}_u$ solves

$$G(p\,|\,n,x) = \alpha/2,\ 0 \leq x < n, \tag{4}$$

cf. Bickel/Doksum [1], pp. 180–2. An explicit solution exists in exactly two cases: If $x = n$ we have $G(p\,|\,n,n-1) = 1-p^n = 1-\alpha/2$ and consequently the confidence interval $[\hat{p}_l, 1] = [(\alpha/2)^{1/n}, 1]$. The case $x = 0$ yields the confidence interval $[0, \hat{p}_u] = [0, 1-(\alpha/2)^{1/n}]$. All other cases call for application of numerical methods.

To solve the equations (2) and (4) we could resort to Newton-Raphson iteration which however makes it necessary to program the derivative of the function under consideration. Instead, we use the Illinois method which is a modification of *regula falsi*, see Ralston/Rabinowitz [6], pp. 338–41. Assume that f is a continuous function defined on an interval $[x_1, x_2]$ such that $f(x_1)f(x_2) < 0$. By the intermediate value theorem of calculus there exists an $x_0 \in (x_1,x_2)$ such that $f(x_0)=0$. This value is unique if f is strictly increasing or decreasing. Regula falsi is a sequence (x_i) of values defined by

$$x_i = x_{i-2} - \frac{y_{i-2}(x_{i-1}-x_{i-2})}{y_{i-1}-y_{i-2}}, \quad i=3,4,5,\ldots \tag{5}$$

where $y_i = f(x_i)$. Notice that x_{i+1} is computed from (x_i,x_{i-1}) if $y_i y_{i-1} < 0$ otherwise it is computed from (x_i,x_{i-2}). This latter case can cause poor convergence and the Illinois method is a modification of this procedure by replacing (x_{i-2},y_{i-2}) for $(x_{i-3},y_{i-3}/2)$ in (5) whenever $y_i y_{i-1} > 0$. This algorithm is implemented in the function `∆x∆←∆a∆ ILLINOIS ∆f∆` where `∆a∆` ↔ x_1,x_2 and `∆f∆` is string describing $f(x)$ in terms of `X` ↔ x. The local `∆eps∆` controls the accuracy of `∆x∆` and can be modified as desired.

```
    ∇∆x∆←∆a∆ ILLINOIS ∆f∆;X;∆y∆;∆eps∆
[1]  ∆eps∆←0.00001
[2]  X←∆a∆[1] ◊ ∆x∆←⍎∆f∆
[3]  X←∆a∆[2] ◊ ∆x∆←∆x∆,⍎∆f∆
[4]
[5]  REGFA:
[6]  →(∆eps∆>|∆x∆[2])/STOP
[7]  X←∆a∆[1]-∆x∆[1]×(-/∆a∆)÷-/∆x∆
[8]  →(0<∆x∆[2]×∆y∆←⍎∆f∆)/ILLI
[9]  ∆a∆←∆a∆[2],X ◊ ∆x∆←∆x∆[2],∆y∆
[10] →REGFA
[11]
[12] ILLI:
[13] ∆a∆←∆a∆[1],X
[14] ∆x∆←(0.5×∆x∆[1]),∆y∆
[15] →REGFA
[16]
[17] STOP:∆x∆←¯1↑X
    ∇
```

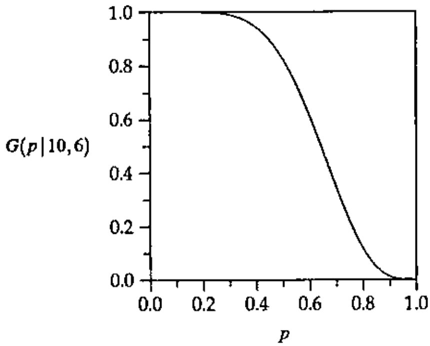
Examples

Example 1. (Ralston/Rabinowitz [6], Ex. 8.3.7(a), p. 401) Compute the root of $f(x) = \cos x - xe^x$. Using `∆eps∆←1E¯10` and `⎕←` in line `ILLINOIS[7]` we get

```
      ⎕PP←10 ⋄ X0←0 1 ILLINOIS '(2○X)-X×*X'
0.3146653378
0.4467281446
0.5338558719
0.5167740785
0.5177442058
0.5177701487
0.5177573635
0.5177573637
      (2○X0)-X0×*X0
4.440892099E¯15
```

Example 2. Arguing for $\hat{p}_u$ it is not hard to see that $G(p|n,x)$ is strictly monotonically decreasing in p for $0 < x < n$ and that $G(0|n,x) = 1$ and $G(1|n,x) = 0$. Hence there exists a unique solution of (4), that is $G(p|n,x) = \alpha/2$. The following expressions illustrate the combination $(n,x) = (10,6)$:

```
      P←0.01×0,⍳100 ⋄ COEFF←(I←0,⍳6)!10
      PLOT P,[1.1] ((P∘.*I)×(1-P)∘.*10-I)+.×COEFF
```



Equipped with the function *ILLINOIS* we have a tool to compute Clopper-Pearson intervals. All it takes is to compute $G(p|n,x)$ from equation (3). This is done in the function *G_BINOM* which makes use of the recursion $P(X=0) = (1-p)^n$ and

$$P(X=x)=\frac{(n-x+1)p}{x(1-p)}\times P(X=x-1),\quad x=1,2,\ldots,n.$$

```
    ∇R←NX G_BINOM P;N;X;Q
[1] N←NX[1] ◊ X←⍳NX[2]  ◊ Q←1-P
[2] ⍎(P=1)/'R←0 ◊ →0'
[3] R←+/×\(Q*N),(P×N+1-X)÷Q×X
    ∇
```

Example 3. The data matrix *WEIGHTS* contains weights before – after (in kg) of young girls receiving a treatment (family therapy) for anorexia. These are taken from Hand et al. [3], p. 229. Notice that the unit kilogram is presumably incorrect because otherwise these data are more typical for girls suffering from *obesity*. Adrian Smith commented that the unit *lbs* seems more appropriate. Nevertheless, we use them to illustrate the application of the Clopper-Pearson method. Let p be the probability that the therapy is a success which means that it will lead to a weight increase. If we assume independence then the number X of successes follows a Binomial distribution with $n=17$ ↔ `1↑⍴WEIGHTS`. From the following expression we see that there have been 13 weight increases so that $(X=13)$.

```
      WEIGHTS,(--/WEIGHTS),[1.1]0>-/WEIGHTS
83.8   95.2 11.4 1
83.3   94.3 11   1
86     91.5  5.5 1
82.5   91.9  9.4 1
86.7  100.3 13.6 1
79.6   76.7 ¯2.9 0
76.9   76.8 ¯0.1 0
94.2  101.6  7.4 1
73.4   94.9 21.5 1
80.5   75.2 ¯5.3 0
81.6   77.8 ¯3.8 0
82.1   95.5 13.4 1
77.6   90.7 13.1 1
83.5   92.5  9   1
89.9   93.8  3.9 1
86     91.7  5.7 1
87.3   98   10.7 1
```

To compute a 95% confidence interval for p enter

```
      0 1 ILLINOIS'0.975-17 12 G_BINOM X'
0.50101
      0 1 ILLINOIS'0.025-17 13 G_BINOM X'
0.93189
```

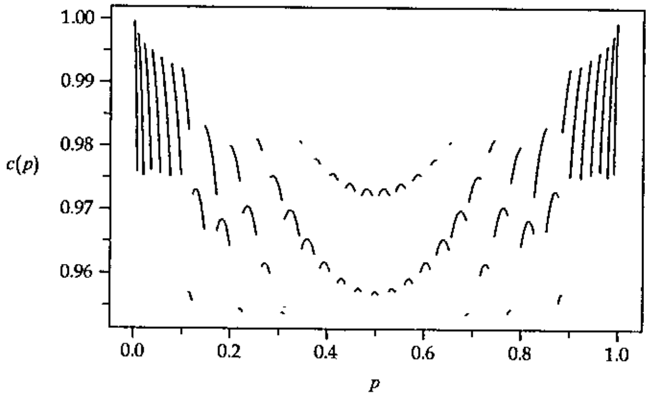
Hence the Clopper-Pearson interval is [0.501,0.932] and we conclude that the therapy is effective.

Example 4. Suppose we want to compute *all* confidence intervals $[\hat{p}_l(x), \hat{p}_u(x)]$ for $x = 0,1,2,\ldots,n$ for a given n. Then it is only necessary to compute the first $m = 1+[n/2]$ intervals $[\hat{p}_l(x), \hat{p}_u(x)], x = 0,1,2,\ldots,m$. Roughly speaking this is because the first intervals for $p \leq 0.5$ can be regarded as a reflection of those for $p \geq 0.5$ To illustrate this point we consider the case *N* ↔ $n = 10$ and $1-\alpha = 0.95$. The variable ***HALF*** consists of the first $m = 1+[n/2] = 1+[10/2] = 6$ Clopper-Pearson intervals. The rest is computed as follows:

```
      (0,⍳N),FULL←HALF,[1]1-⊖⌽((⌈0.5×N),2)↑HALF
  0 0         0.3085
  1 0.0025286 0.44502
  2 0.02521   0.5561
  3 0.066739  0.65245
  4 0.12155   0.73762
  5 0.18709   0.81291
  6 0.26238   0.87845
  7 0.34755   0.93326
  8 0.4439    0.97479
  9 0.55498   0.99747
 10 0.6915    1
```

Example 5. To demonstrate formula (1) suppose we take $n = 30$ and $1-\alpha = 0.95$. After having computed all 31 intervals $[\hat{p}_l, \hat{p}_u]$ it is relatively easy to compute the coverage probability $c(p) := P(\hat{p}_l \leq p \leq \hat{p}_u)$ using a program ***COVERAGES_BINOM*** (not displayed here). Its output can be used to plot $c(p)$ against p. Notice that $c(p) \geq 0.95$ holds throughout:

Coverage probabilities $c(p)$ for CP-intervals, $n = 30$



Concluding Remarks

Using the Illinois method it is easy to compute all Clopper-Pearson intervals for the parameter p of a Bernoulli distribution as no derivatives have to be computed. In this way one avoids the tedious and inexact task of estimating the bounds from charts. For large sample sizes one can resort to an approximation using deMoivre-Laplace's version of the Central Limit Theorem.

Notice that formulas (2) and (4) can be applied to compute lower or upper bounds for p if one replaces $\alpha/2$ by α. For instance, for the data from Example 3 to compute a 95% lower bound for p enter

```
      0 1 ILLINOIS '0.95-17 12 G_BINOM X'
0.53945
```

Finally, it should be obvious that the approach can be easily extended to other discrete distributions such as the Poisson or negative binomial.

Acknowledgement

I would like to thank Bernhard Strohmeier who helped in LaTeXing this paper.

References

[1] Bickel, P.J./Doksum, K. (1977). *Mathematical Statistics: Basic Ideas and Selected Topics*, Holden-Day, San Francisco.

[2] Clopper, C.J. / Pearson, E.S. (1934). "The Use of Confidence or Fiducial Limits Illustrated in the Case of the Binomial", *Biometrika* **26**, 404–413.

[3] Hand, D.J. / Daly, F. / Lunn, A.D. / McConway, K.J. / Ostrowski, E. (1994). *A Handbook of Small Data Sets*, Chapman & Hall, London.

[4] Mood, A.M. / Graybill, F.A. / Boes, D.C. (1974). *Introduction to the Theory of Statistics*, 3rd edition, McGraw-Hill, New York.

[5] Pearson, E.S. / Hartley, H.O. (1963). *Biometrika Tables for Statisticians*, volume 1, 2nd edition, Cambridge University Press, Cambridge, England.

[6] Ralston, A. / Rabinowitz, P. (1978). *A First Course in Numerical Analysis*, 2nd edition, McGraw-Hill, Tokyo.

The Incomplete Elliptic Integrals and APL

Joseph De Kerf

In a previous note [1], it was shown how the *common mean* concept [2] may be used to calculate the complete elliptic integrals of the first and second kind [3, 4], illustrated by user-defined functions in APL. In this third note of the series, it is shown how the concept may be used to calculate the incomplete elliptic integrals of the first and second kind [3, 4]. APL user-defined functions are given.

The incomplete elliptic integrals of the first kind $K(\phi;p)$ and of the second kind $E(\phi;p)$ may be defined as:

$$K(\phi;p)=\int_0^{\phi}\frac{d\theta}{\sqrt{1-p^2\sin^2\theta}}$$

$$E(\phi;p)=\int_0^{\phi}\sqrt{1-p^2\sin^2\theta}\quad d\theta$$

with $p^2\leq 1$ or $-1\leq p\leq 1$. Both incomplete elliptic integrals are even functions of p:

$$K(\phi;-p)=K(\phi;p)\ \text{ and }\ E(\phi;-p)=E(\phi;p)$$

In addition:

$$K(-\phi;p)=-K(\phi;p) \qquad\qquad K(\Pi/2;p)=K(p)$$
and, as special cases
$$E(-\phi;p)=-E(\phi;p) \qquad\qquad E(\Pi/2;p)=E(p)$$

the complete elliptic integrals of the first and second kind respectively.

The incomplete elliptic integrals may be calculated for instance by numerical integration or series expansion [3, 4]. Once again however, much easier to program — and this is especially true in APL — are the algorithms based on the common mean concept [3, 4].

Indeed, the incomplete elliptic integrals of the first kind $K(\phi;p)$ may be calculated by application of the relation:

$$K(\phi;p)=\frac{\phi_\infty}{c(1,q)}$$

where $c(1,q)$ is the common mean of 1 and $q=\sqrt{1-p^2}$

and $\phi_0=\phi$

$$\phi_{i+1}=\phi_i-\frac{1}{2^{i+1}}\arctan\left(\frac{(x_i-y_i)\tan(2^i\phi_i)}{x_i+y_i\tan^2(2^i\phi_i)}\right)$$

$$=\phi_i-\frac{1}{2^{i+1}}\arctan\left(\frac{(x_i-y_i)\sin(2^{i+1}\phi_i)}{2(x_i\cos^2(2^i\phi_i)+y_i\sin^2(2^i\phi_i))}\right)$$

the x_i 's and y_i 's being the successive arithmetic and geometric means respectively, while calculating the common mean $c(1,q)$. The sequence ϕ_i converges toward ϕ_∞ as rapidly as the x_i 's and y_i 's toward their common mean $c(1,q)$.

A user-defined function *IEI*1, for $-\Pi/2\le\phi\le\Pi/2$, based on this procedure, with as left argument ϕ and as right argument p, may be:

```
      ∇IEI1[⎕]∇
     ∇R←F IEI1 P;I;A;B
[1]   →((1≠|P)∨(○÷2)=|F)/LAB1
[2]   →0,R←⍟3○0.5×F+○0.5
[3]  LAB1:R←(2+I←¯1),(1-P*2)*0.5
[4]  LAB2:A←(-/R)×1○F×2*1+I←I+1
[5]   B←2×+/R×(2 1○F×2*I)*2
[6]   F←F-(¯3○A÷B)÷2*I+1
[7]   →(≠/R←(0.5×+/R),(×/R)*0.5)/LAB2
[8]   R←F÷0.5×+/R
     ∇
```

which for $\phi=\Pi/6$ and $\phi=\Pi/3$, $p=0.0(0.1)1.0$, and comparison tolerance `1E¯15` gives respectively:

```
      R1←(○÷6)IEI1¨P←0,0.1×⍳10
      R2←(○÷3)IEI1¨P
      3 1 20 15 20 15⍕P,R1,[1.5]R2
0.0   0.523598775598299   1.047197551196598
0.1   0.523825500165390   1.048738631962169
0.2   0.524508805294440   1.053430587029900
0.3   0.525658228737263   1.061489706726052
0.4   0.527290159175087   1.073313629047138
```

```
0.5     0.529428627051906      1.089550670051885
0.6     0.532106525784461      1.111233322932336
0.7     0.535367402759971      1.140044752769332
0.8     0.539268044090846      1.178902299538824
0.9     0.543882214161571      1.233446325452344
1.0     0.549306144334055      1.316957896924817
```

For $\phi = \Pi/2$, p = 0.0(0.1)0.9 and comparison tolerance `1E¯15` the user-defined function *IEI*1 gives:

```
        R←(○÷2)IEI1¨P←0,0.1×⍳9
        3 1 20 15⍕P,[1.5]R
0.0     1.570796326794897
0.1     1.574745561517356
0.2     1.586867847454166
0.3     1.608048619930513
0.4     1.639999865864511
0.5     1.685750354812596
0.6     1.750753802915752
0.7     1.845693998374723
0.8     1.995302777664729
0.9     2.280549138422770
```

while for $\phi = \pm\Pi/2$ and p = ±1, as the integral is infinite (∞), a domain error is reported and the user-defined function is suspended. Eventually, the error report may be trapped and an appropriate response may be programmed.

As for $\phi = \Pi/2$ the elliptic integral of the first kind $K(\phi;p)$ is the complete elliptic integral of the first kind $K(p)$, the results are the same as those reported for CEI1 in (1).

It should be noted that for $\phi \neq \pm\Pi/2$ and $p = \pm 1$, both the limit of ϕ_i and the common mean of $c(1,q)$ are zero, and the procedure fails. However, the integral may be evaluated analytically and as such, in lines 1 and 2 of the user-defined function *IEI*1, the result R is explicitly set to its value $\ln\tan(\phi/2 + \Pi/4)$. This cannot be done in one line because then, for $\phi = -\Pi/2$ a domain error would be reported, whether $p = \pm 1$ or not. During the loop, in lines 4 and 5, the local variables A and B are introduced to improve readability, while the overhead is rather small.

The incomplete elliptic integral of the second kind $E(\phi;p)$ may be calculated by application of the relation:

$$E(\phi;p)=\frac{\phi_\infty}{2.c(1,q)}\left(2-\sum_{i=0}^{\infty}2^i\left(x_i^2-y_i^2\right)\right)+\frac{1}{2}\sum_{i=0}^{\infty}(x_i-y_i)\sin\left(2^{i+1}\phi_{i+1}\right)$$

or $$E(\phi;p)=\frac{\phi_\infty}{2.c(1,q)}(2-S)+\frac{T}{2}$$

and $$S=\sum_{i=0}^{\infty}2^i\left(x_i^2-y_i^2\right)$$

$$T=\sum_{i=0}^{\infty}(x_i-y_i)\sin\left(2^{i+1}\phi_{i+1}\right)$$

where $c=(1,q)$, the ϕ_i's, and the x_i's and y_i's are defined as for $K=(\phi;p)$.

A user-defined function *IEI2*, for $-\Pi/2\le\phi\le\Pi/2$, based on this procedure, with as left argument ϕ and as right argument p, may be:

```
      ∇IEI2[⎕]∇
     ∇R←F IEI2 P;I;S;T;A;B
[1]    →(1=|P)/0,R←1○F
[2]    R←(1+S←T←1+I←¯1),(1-P*2)*0.5
[3]   LAB:A←(-/R)×1○F×2*1+I←I+1
[4]    B←2×+/R×(2 1○F×2*I)*2
[5]    F←F-(¯3○A÷B)÷2*I+1
[6]    S←S+(-/R*2)×2*I
[7]    T←T+(-/R)×1○F×2*I+1
[8]    →(≠/R←(0.5×+/R),(×/R)*0.5)/LAB
[9]    R←(T÷2)+(2-S)×F÷+/R
     ∇
```

which for $\phi=\Pi/6, \Pi/3, \Pi/2$, $p=0.0(0.1)1.0$, and comparison tolerance `1E¯15` gives respectively:

```
      R1←(○÷6)IEI2¨ P←0,0.1×⍳10
      R2←(○÷3)IEI2¨ P
      R3←(○÷2)IEI2¨ P
      3 1 19 15 19 15 19 15⍕P,R1,R2,[1.5]R3
0.0   0.523598775598299  1.047197551196598  1.570796326794897
0.1   0.523372224005088  1.045660219705633  1.566861942021668
0.2   0.522691528560574  1.041025536968957  1.554968546242529
```

0.3	0.521553538774118	1.033223451406541	1.534833464923249
0.4	0.519952906838045	1.022130132787699	1.505941612360040
0.5	0.517881934859938	1.007555555144472	1.467462209339427
0.6	0.515330345384322	0.989221593549378	1.418083394448724
0.7	0.512284956933147	0.966723133094528	1.355661135571956
0.8	0.508729236545024	0.939454803724951	1.276349943169906
0.9	0.504642686598563	0.906456986263154	1.171697052781614
1.0	0.500000000000000	0.866025403784439	1.000000000000000

As for $\phi = \Pi/2$ the elliptic integral of the second kind $E=(\phi;p)$ is the complete elliptic integral of the second kind $E(p)$, the results are the same as those reported for `CEI2` in [1].

It should be noticed that for $p = \pm 1$, both the limit of ϕ_i and the common mean $c=(1,q)$ are zero, and the procedure fails. However, the integral may be evaluated analytically and as such, in line 1 of the user-defined function `IEI2`, the result R is explicitly set to its value $\sin(\phi)$. During the loop, in lines 3 and 4, the local variables A and B are introduced to improve readability, while the overhead is rather small. The local variables S and T however, cannot be omitted, as their values are needed in the final evaluation of $E(e;p)$ in line 9.

For both user-defined functions, the relative accuracy of the common means calculated is at least the current value of the comparison tolerance, for the results shown `1E¯15`. In practice, however, this relative accuracy is slightly better, as the exact value of this common mean lies between the two iterations calculated and the result is set to the arithmetic mean of the final values of those iterations.

Note: the number of iterations needed and the cpu time depend on the current value of the comparison tolerance and on p, but don't depend on ϕ. Results of a benchmark for `1E¯15` and p = 0.1(0.2)0.9 are shown below and compared with those for `CEI1` and `CEI2`. The cpu times are in milliseconds.

P	0.1	0.3	0.5	0.7	0.9
number of iterations	3	3	4	4	5
CEI1 P	112	112	138	138	164
(○÷2) IEI1 P	347	347	441	441	536
CEI2 P	209	209	256	256	303
(○÷2) IEI2 P	478	478	612	612	746

Of course, the user-defined functions *IEI*1 and *IEI*2 are considerably slower than *CEI*1 and *CEI*2 respectively. This means that, when the user in an application only needs the complete elliptic integrals, it is much more efficient to refer to the user-defined functions *CEI*1 and *CEI*2.

The calculations and the benchmark were done on a MicroVAX 2000, with VAX APL Version 3.1, under VMS Version 4.7 [5]. The default value for the comparison tolerance in VAX APL is 1E¯15. The number of iterations has been measured by globalizing the counter I. The cpu times have been measured with the system function ⎕*MONITOR*.

References

[1] J. De Kerf, *The Complete Elliptic Integrals and APL*, Vector 12.1, pp 102–105, 1995

[2] J. De Kerf, *The Common Mean and APL*, Vector 11.3, pp 21–23, 1995

[3] M. Abramowitz and I.A. Stegun (Eds), *Handbook of Mathematical Functions*, Applied Mathematics Series no. 55. National Bureau of Standards, Washington D.C., 1964 (and subsequent revisions)

[4] J. Spanier and K. B. Oldham, *An Atlas of Functions*, Hemisphere Publishing Corporation, New York, 1987

[5] *VAX APL Reference Manual – Vols 1 and 2 (Software Version V.3.0)*, Publication Numbers: AI-P142D-TE and AI-GV09B-TE. Digital Equipment Corporation, Maynard, Mass., June 1987

Joseph De Kerf
Rooienberg 72
B-2570 Duffel
Belgium

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: Gremlins, Pixels and Brownie Points

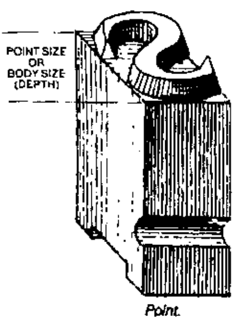
by Ray Cannon

I thought I might have gremlins in a Dyalog APL/W system I recently produced. A letter printed on A4 paper using a standard True Type (TT) font (Times New Roman) would be printed at varying sizes, even though the same font size was requested. This was not winning me any Brownie points.

As a test I would produce a standard printout using Windows WRITE.EXE selecting the Times font with a size of 36 points (approximately ½ inch high characters) as a control, and then print the same output from Dyalog and compare the results.

Upon investigation, this behaviour appeared to be related to the printer driver being used (although now I have found an example where I can produce this behaviour by varying the printer setup for handling TT fonts between downloading as bit image and downloading as graphics).

The problem (and solution) was finally revealed after spending a couple of hours with John Daintree of Dyadic Systems Ltd.



(from *"Basic Typography — a Design Manual"* by James Craig. Published by Watson-Guptill, New York, 1990)

There are two (common) ways of specifying the height of a font: you can specify the "cell height" or the "character height". To explain the difference, think back to an old-fashioned printing press with lead type. To vary the gap between lines of type, a lead spacer is placed between them. This spacer is called "leading" (as in the metal). In the simple case, the cell height is equal to the character height plus the "leading" height. It is possible to produce fonts with "built in" leading, useful if you want to have line-drawing characters able to produce continuous vertical lines (that is to say the vertical bar character extends to the full extent of the character cell).

Under Windows, True Type fonts' point sizes specify the "character height" not the "cell height". The parameter specifying the font size that is passed to the utility that creates a (logical) font within Windows can define either the "cell height" or the "character height". This is done by using a positive number of pixels for the "cell height" or a negative number of pixels for the "character height". Under Dyalog, the Size property of the Font object supports this protocol. (When you interrogate the Size of a Font, Dyalog always returns the positive "cell height".)

Armed with this knowledge, I tested a dozen or so printer drivers (there is no need to have the physical printer attached) creating fonts specifying the size in both positive and negative pixel sizes, and then reading the size returned. The results are shown below (Table 1). From this table I surmise that the way the printer driver handles TT fonts determines how it calculates the "leading" height. In particular when the TT font is downloaded as a bit image, the "leading" height is zero.

The difference between the cell height and the character height is the internal leading, as demonstrated by the following diagram:

```
----------   <-------------------------------
|        |            |- Internal Leading   |
| |    | |   <---------                     |
| |    | |          |                       |- Cell Height
| |----| |          |- Character Height     |
| |    | |          |                       |
| |    | |          |                       |
----------   <-------------------------------
```

```
     ∇ r←device Points2Pixels pointsize;res
   ⍝ Returns the font "Size" property value required to generate
   ⍝ a True Type font with the specified point size on the
   ⍝ specified device.

   ⍝ Calculate the resolution of "device"
    res←÷/⊃¯2↑device ⎕WG'DevCaps' ⍝  pixels per mm
    res←25.4×res ⍝                   pixels per inch
    res←res÷72 ⍝                     pixels per point
    r←pointsize×res ⍝                pixels for point size
   ⍝ Make negative as TT point size is Character height
   ⍝  not Cell height, and round to integer
    r←-⌊0.5+r

   ⍝ NOTE Some devices such as screens, work in "logical units"
   ⍝ (e.g. logical inch) rather then physical units (inches) under
   ⍝ MS Windows to cater for their low resolution.
  ∇
```

Table 1

Driver	Cell Height of font returned on requesting creation of font size	
(default settings)	50 pixels	-50 pixels
HP LJ II	50	57
HP LJ III	50	50
HP LJ III Postscript	50	59
HP LJ II Postscript	50	59
HP LJ IV	50	57
HP DeskJet 120C	50	57
Generic Postscript	50	59
Epson FX 80	50	59
Canon BJ10e	50	57
Xerox 4045	50	50
IBM ProtoPrint	50	50
Brother HL-6V		
Download as bit image	50	50
Download as graphics	50	57

Note that ALL the fonts created with a size of -50 look the same size. Those created with a size of +50 are of varying size.

Addendum (from Adrian Smith)

This solves a considerable puzzle for me — in particular I have always had the problem with my RAIN graphics printing that the font size was not predictable across printers. The basic message is very simple — use -ve sizes always, and what you get will be what you want. Thanks Ray!

TECHNICAL CORRESPONDENCE

APL-Fortran Calls using quadNA

APL2/OS2 ← ⎕NA → FORTRAN

From: Professor R.G. Selfridge September 95

Many APLs in recent years have provided facilities to connect across to compiled software in some other language. I have looked at the process in APL2 on PCs, in particular under the latest version, APL2/OS2.

Mainframe APL provides ⎕*NA*, or 'name association', along with a file that must describe the elements in the calling sequence. Naturally there are variations to fit the ultimate language that will be 'associated'. The first version of APL2 for PCs was constructed for DOS-based hardware, and while it had a connection across to compiled routines and followed many of the constructs of the mainframe version any resemblance was superficial. An attached support processor was used (with the usual calling connections). The Fortran source (as a subroutine) was compiled and linked, with one or more entry points. Some provided APL functions then took the appropriate listings and generated source for an assembler that was then used by a supplied PFORTPAR.EXE file to complete the connections. While it sounds complicated, if the instructions were followed closely the result usually worked. The assembler source contained data about each variable in the calling sequence, type, size, results returned, as might be expected. Usage now required connecting to the attached support processor, sharing the needed two variables and then sending data over and bringing back results.

The arrival of APL2/2 opened up this connection, and materially simplified its usage. ⎕*NA* was now the interface, a file contained all the data about variables in the calling sequence, and the actual compiled Fortran code was stored as a `***.DLL` file. The conventions for the identifying file become very similar to that of the mainframe, i.e. type, dimensions of each variable in the calling sequence, with a flag for 'returning results'.

It is, however, reasonably clear that the principal use of this connection, within the development group, was to connect to C or C++ programs, and Fortran had not been used (this is not meant as any condemnation, just a statement of apparent fact). Fortran has some problems which can be easily overcome, once you have the 'scheme of things'. Here they are.

1. APL2/2 runs under OS2, version 2 or later. This is a 32-bit addressing system. As of late 1994 there was only one Fortran that was built around 32-bit addressing (as far as I or the APL development group could discover). IBM's earlier Fortran (ex Microsoft) would compile and link under OS2, but all Real*8 operations appear to give 0 as an answer. There are several Fortrans that have extenders to access higher memory, but these also probably have trouble in the OS2, v.2 environment (I have not actually tested them, no promises). The Fortran that was used came from Watcom, Fortran77(32) (for the rest of this article I shall just refer to it as Fortran). This Fortran has the ability to compile and link into an ***.DLL (Dynamic Link Library) subject to a few caveats.

2. Any floating point number in APL2 is based on a high precision representation, whether as REAL or COMPLEX. The ⎕*NA* association will provide for matching in these conversions. As a result most Fortran source that is intended for sharing will have been built around declarations of REAL*8 and COMPLEX*16. All Fortrans provide for 'generic' functions for many functions; Watcom 77 is no exception. There are two exceptions, which are documented if the user thinks to look for them (I did not at first). Here they are:

   If X is COMPLEX*16 and you want to get the real part you must use DREAL(X). The choice of REAL(X) will give only single precision. (Surprise, IMAG works as you might expect).

   If X and Y are REAL*8, the double precision complex requires DCMPLX(X,Y). If you use CMPLX(X,Y) you will get a single precision complex.

3. In order to use ⎕*NA* the calling sequences must follow OS2 conventions exactly. Hence there can be no communications that use registers. When staying entirely in a Fortran environment, Watcom allows for register communicating; APL2 cannot. Thus you must force the compiler to avoid registers in communications, and thus the compiler must be given two options, /BD which says a DLL is being built, and /SC which effectively says not to use registers in communicating.

4. There are two files that are needed for ⎕*NA*, one of which describes the contents of the calling sequence, the other is the ***.DLL file just created. These must be put in the proper libraries. While many APLers may know where these are, or can interpret what 'default' really means, the documenting is 'a wee tad short' in this respect. Add to that that the OS2 'search and find' utility first located a back-up directory when I went searching, and it is not surprising that nothing really worked. The calling sequence file, which might typically be inside an ***.NAM file, should be stored inside the BIN sub-directory of the directory APL2OS2. There is a sample .NAM file provided, but it is suggested that the user not add to that file, it could be changed in a later service release. The translated Fortran should be stored as an ***.DLL file inside the DLL sub-directory of the directory APL2OS2.

Example

Suppose we have a Fortran routine ROOTS(N,X,Y), which takes the complex coefficients X of a polynomial of degree N, and returns the solutions in Y. We assume the source has been sufficiently well checked that errors are remote. Here are the steps.

1. **Compile the source:**
   C:\ wfc386 /bd /sc ROOTS

   /bd forces a .DLL design, /sc controls registers in the calling sequences. There may be several 'entry points', but there is only one in this example (more entry points won't affect this command).

2. **Link.** While this can be done with a 'linker directive' file, I prefer (upper case is from system, except for control z).

   C:\ wlink

WLINK system os2v2 dll	for OS2, as a DLL file
WLINK file roots	assumes there is a file ROOTS.OBJ
WLINK export roots	entry point is roots, as many entries as desired, each entry should be 'export'ed
WLINK CTL Z	ends linker entries

   Your directory will now contain ROOTS.OBJ from the compilation (no need to keep), and a file ROOTS.DLL which needs to be moved to the sub-directory DLL of the directory APL2OS2.

3. **Create the communications data lists.** This is well described in available documention. For this example X is complex, of size determined on use, as is Y. Since Y is used to return results it must be created properly in the APL using routine (see later). The file, called TEST.NAM, is stored in the BIN sub-directory of the directory APL2OS2, and is

```
:nick.ROOTS
:link.SYSTEM
:lib.ROOTS
:proc.ROOTS
:valence 0 1 0
:rarg(G4 1 3)(I4 0)(J16 1 *)(>J16 1 *)
```

The usage of > in the last line is to indicate that this argument is used for returning results. There is some case sensitivity, so create the Fortran routine with upper case for the routine name, and keep all names upper case in TEST.NAM. The APL program that calls this routine is now (shortened to show only the critical parts)

```
Y ←ROOTS X;N;ROOTS
N ←'<TEST.NAM>' 11 ⎕NA'ROOTS'
⍝Comment:TEST.NAM is name of characteristic file
Y ←(N←(-1)+↑ρX)ρ 0J1
⍝Comment:  create Y of correct length and type
ROOTS N W 'Y'
```

Comment: The argument for *ROOTS* must be a nested list, so no " , ".

Since results are in *Y*, it is passed as a name. *N* is generated as a scalar with ↑ since it is defined as a scalar in TEST.NAM. The results will be in the variable *Y*. I note that since the DLL file has the entry name ROOTS, it is important to make *ROOTS* local to the APL function. If enough different names are used this localizing can be avoided.

The primary use of ⎕*NA* that I have made is this one application, but I believe it should cover nearly all uses of Fortran. I have not attempted using a Fortran function routine (i.e. one that transmits a result back in-line) with ⎕*NA*.

I also offer many thanks to the APL2 development group, who offered major support in tracking all this down.

Ralph Selfridge
300 CSE
University of Florida
Gainesville, FL 32611, USA
Email: selfrid@nervm.nerdc.ufl.edu

Calculating nCr

From: Norman Thomson 28th August 1995

Alan Sykes rightly makes the point in *Calculating Probabilities for Elementary Distributions* (Vector Vol.12 No.1) that anyone who is in the business of doing serious probability calculations should regard a function *LNNCR* which computes the logarithm of **nCr** as essential. The function given by Alan can be appreciably shortened by using "each". First a basic function for scalar *n* and *r* is:

```
[0]   Z←n lnncr r
[1]   Z←+/(⍟1+n-⍳r)-⍟⍳r←r⌊n-r
```

which can be generalised to Alan's *LNNCR*, where the right argument is an array of any shape, by:

```
[0]   Z←N LNNCR R
[1]   Z←(⍴R)⍴N lnncr¨,R
```

If only a small number of values are required, which is often likely to be the case in practice, this is the simplest way to go about things.

However, as Alan points out, if the right argument is an array with a largish number of items, an efficiency gain is achieved by computing many **nCr**s, and then selecting the ones which are actually wanted. An extension of lnncr to do this is

```
[0]   Z←n lnncr1 r;t;u
[1]   Z←0,+\(⍟1+n-⍳u)-⍟⍳u←⌈/t←u⌊n-u←,r
[2]   Z←(⍴r)⍴Z[1+t]
```

Alan says that extending this to array left arguments as for the "shriek" primitive is tricky and calls for a volunteer. I suggest:

```
[0]   Z←n LNNCR1 r
[1]   Z←n lnncr1¨⊂r
```

Unless Alan has some subtlety in mind which is escaping me, it seems I have just volunteered! Turning to *HYPERGEOM*, the function which evaluates hypergeometric probabilities, I prefer the form:

```
[0]   Z←r HYPERGEOM Nnm;N;n;m
[1]   (N n m)←Nnm
[2]   Z←*-/(n,N-0,n)lnncr¨r,m-0,r
```

This is not only shorter than Alan's version, but also emphasises the essential symmetry of the calculation, which is not apparent in the mathematical notation, namely that the *n,N-0,n* which makes up the left argument to `lnncr` refer to the overall sizes of population and sample, while *r,m-0,r* refer to the respective numbers of "successes".

Continuing Alan's illustration of the National Lottery odds, the inverse probabilities of gaining 3, 4, 5, 6 successes, that is the odds expressed as "1 in ..." are given by

```
      ⌈÷3 4 5 6 HYPERGEOM¨⊂49 6 6
57 1033 54201 13983816
```

Next, Poisson probabilities and cumulative probabilities are given by the following chains of functions:

```
[0]   Z←N pois MU
[1]   Z←*(+/(⍟MU)-⍟⍳N)-MU

[0]   Z←N pois1 MU
[1]   Z←N pois¨MU

[0]   Z←N POIS MU
[1]   Z←(⊂N)pois1¨MU

       8 4⍕⊃0 1 2 3 4 5 POIS 1 2
   .3679   .3679   .1839   .0613   .0153   .0031
   .1353   .2707   .2707   .1804   .0902   .0361

[0]   Z←N poiscdf MU
[1]   Z←+/((↑N)+0,⍳-/-N)pois¨MU

[0]   Z←N poiscdf1 MU
[1]   Z←+\((0,1↓¯1⌽1+N),¨N)poiscdf¨MU

[0]   Z←N POISCDF MU
[1]   Z←(⊂N)poiscdf1¨MU
       1000 POISCDF 1000
 0.50840937
       8 4⍕⊃100 120 140 POISCDF 100 120
    .5266   .9773   .9999
    .0347   .5243   .9669
```

The functions `LNNCR1`, `POIS` and `POISCDF` all work with lower rank arguments, so that the subsidiary functions are not required at user level.

Sooner or later, as the size of the integer arguments increases, even the above functions will hit limits, either `domain errors`, if even the logarithms exceed the largest representable number, or *vsfull* if the number of integers in the numerator and denominator of the recursive formula become too large.

The first problem is addressed by increasing the base value of the logarithms (the value *e* is, after all, arbitrary).

The second problem is addressed by splitting the recursively defined fraction into blocks of a size which nearly fills the workspace, and using iteration. (On my computer 10000 is a desirable size.)

When computing very large binomial coefficients, it might well be the case that the number of decimal digits in the result was the primary object of interest, in which case it is appropriate to use:

```
[0]  Z←n lnncr10 r
[1]  Z←+/(10⍟1+n-⍳r)-10⍟⍳r←r⌊n-r
```

and the iterative function indicated above is:

```
[0]  Z←n lcr r;t;u;i
[1]  Z←n lnncr10 10000⌊ t←r⌊n-r ⋄ i←0
[2]  L1:→(t<u←10000×i←i+1)/0
[3]  Z←Z++/(10⍟1+n-u)-10⍟u+⍳10000 ⋄ →L1
```

As an example, the number of decimal digits in **1000000 C 500000** is:

```
      ⌈1000000 lcr 500000
302464
```

Floating-Point Precision

From: Joseph De Kerf 23 August 1995

In [1], it was shown how the so-called common mean concept may be used to calculate the complete elliptic integrals of the first and second kind. APL user-defined functions were given and illustrated for p = 0.0(0.1)0.9/1.0. Results were displayed with 10 and 15 digits after the decimal point. Calculations were done with VAX APL under VMS (DEC) and for comparison tolerance its default value 1E15.

In a companion note however, the production manager observes he found slightly different results when run in Dyalog/W. So I did my homework again, however with comparison tolerance 1E16 (instead of 1E15). In addition I did the same calculations but with APL user-defined functions based on:

1. gaussian quadrature (Newton-Cotes formulae) and
2. series expansion (MacLaurin's series integrated, Addams-Hippisley series and the series dedicated to A. Caley).

For all of these runs, carefully avoiding the accumulation of errors as far as possible, I found the same results as published in [1]. So, I suppose those results are correct.

On the other hand, I did the same calculations on other APL systems and found, for all of them, some differences similar to or even greater than those observed by the production manager.

I suppose that the origin of the difference lies in the floating-point mantissa precision. For VAX APL under VMS this is more than 16 decimal digits (about 16.86D), such that the 16th decimal digit may be correct. However, for all the other APL systems I checked, floating point representation is based on the ANSI/IEEE Std 754-1985 [2]. This means that for those systems the floating-point mantissa precision is somewhat less than 16 decimal digits (about 15.95D), such that the 16th decimal digit of the intermediate and *a fortiori* of the end results becomes more or less unreliable.

This is the reason why, in some implementations such as IBM's APL2/PC and MicroAPL's APL.68000, the upper limit of the domain of the print precision variable is set to 15. On the other hand, the advantage of the standard is that the largest number representable is about 1.80E308, while in practice, who cares about the accuracy of the 16th decimal digit?

Joseph De Kerf
Rooienberg 72
B-2570 Duffel
Belgium

References

[1] Joseph De Kerf; *The Complete Elliptic Integrals and APL;* Vector, Vol 12, No. 1 July 1995, pp 102-106

[2] *ANSI/IEEE Std 754-1985;* IEEE Standard for Binary Floating-Point Arithmetic; Published by the IEEE, New York, USA; 12 August 1985

Microsoft Announces APL for the 8088 - almost 20 years late!

From: Adrian Smith Sept 7th 1995

References from "Gates" by Stephen Manes and Paul Andrews Touchstone, 1993-4

I came across this excessively flabby tome (mostly gossip and hearsay — not recommended for general reading) in Germany last year, and casually looked up APL in the index, as one does. To my surprise, there were several entries and some quite thorough references from the early microcomputer press. Here are the more interesting ones:

Page 91 – "An Open Letter to Hobbyists" from early 1976 ...

> "To me, the most critical thing in the hobby market right now is the lack of good software courses, books and software itself. Without good software and an owner who understands programming, a hobby computer is wasted. Will quality software be written for the home market? ...
>
> ... the fact is, no one besides us has invested a lot of money in hobby software. We have written 6800 BASIC, and are writing 8080 APL and 6800 APL, but there is very little incentive to make this software available to hobbyists."

Pages 97-98

> Later that summer Bill headed off to Seattle to work on a pet project he had casually mentioned in both open letters and the MicroKid ad had described as "Upcoming bout: APL for 8080." APL was the acronym for an offbeat programming language dubbed with terminal feyness "A Programming Language," and Bill had become infatuated with it. Languages often create cults around them, and APL had one of the most vociferous.
>
> An interpreted language, APL was extremely condensed: In a couple of well-thought-out APL statements, you could do what would take line after line of code in other languages. APL used special symbols, its own goofy alphabet: Full of arrows and deltas and squiggles, the code appeared to be written in hieroglyphs or Greek or something. All this gave APL high marks among mathematicians and scientists who liked its elegant, powerful solutions to complex vector and matrix problems. APL code was almost impossible to read and therefore hard for a third party to understand and maintain, but it was popular enough to be one of two languages (BASIC was the other) available

for the IBM 5100, the first in the IBM series of small computers — some would later say personal computers — that eventually would lead to the IBM PC.

A year or so before, Gates had been introduced to the language by Mike Courtney, a Seattle APL specialist who had read about Altair BASIC, sent for the manual, and couldn't believe a version that good had actually been been implemented on a microprocessor. A casual reference to APL in a phone call to Gates led to a meeting that summer in Seattle, where Courtney had fired up Bill's enthusiasm for the language. Over the following year, Gates delved deeper into APL, to the point of examining at least one version of its source code.

At least one person saw Bill's affection for the language as a hopeless crush. Back in Seattle for a visit, Paul Allen lunched with Courtney to discuss the possibility of his coming work with MITS. The two had never met before, and, sensing Allen's coolness, Courtney asked what the problem was. The problem, Allen told him, was that Courtney had Bill working on this APL thing, and it was a bunch of crap. Allen wanted Gates to work on something real, like FORTRAN or COBOL — something they could actually sell.

But Weiland agreed with Bill. FORTRAN and COBOL seemed sort of passé; APL was avante-garde. Now all that was left was to write a version of it for the 8080. In Seattle for the summer of 1976, Gates seemed to be making headway, and by August he was telling the Northwest Computer Club that it ought to be finished in the fall. In the fall Bill returned to Harvard with APL uncompleted. By January the club newsletter was asking "Whatever happened to MicroSoft's 8080 APL?"

Page 124 – dated mid 1975

And by all accounts, Gates's killer schedule and corrosive social skills kept him out of whatever dating scene Alberquerque had to offer. In this era, Microsoft was his only mistress.

It could be a cruel one. Bill's APL continued to exist only in an incomplete state on yellow legal pads in a desk drawer somewhere. .... The lack of APL was hardly a make-or-break issue. Bill Gates really was busy — not just programming, but making contacts with customers. Chipmakers came calling. Intel wanted BASIC. National Semiconductor took BASIC and FORTRAN for its development systems. And COBOL was finally ready.

Given the computing power available, and the complexity of the problem, the outcome was probably inevitable. However it is interesting to speculate how different things might have been if Bill had made it work on the 6800 back in 1976!

At Work and Play with J
The Bauer-Mengelberg Problem

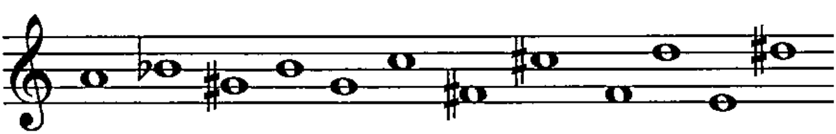
by Eugene McDonnell

This paper discusses a combinatorial problem arising in the field of music, and shows the importance of the `A.` primitive discussed in my last column.

The problem was told to me many years ago by Ken Iverson, who had heard it from Adin Falkoff, who in turn had heard it from Stephen Bauer-Mengelberg, a conductor / programmer who was a colleague of Ken and Adin's at IBM's Systems Research Institute at UN Plaza in New York City in the early 1960s. [Picturesque but irrelevant detail: Adin tells of asking Bauer Mengelberg how one of the pieces he conducted at a concert the night before had gone. The answer was "The first movement went only so-so, but with the second movement I floated off the podium."]

The problem deals with the twelve-tone music associated with the composer Arnold Schoenberg. I am not a musician, so I shall only briefly describe it musically, and then convert it into a problem in combinatorial mathematics.

The problem is to describe all the ways in which the twelve semitones of the octave can be written so that each is used exactly once, and so that each interval possible within the octave occurs exactly once. The Penguin book *A Dictionary of Music*, by Robert Illing (1950) gives an example of such a piece in figure (f) on page 297.



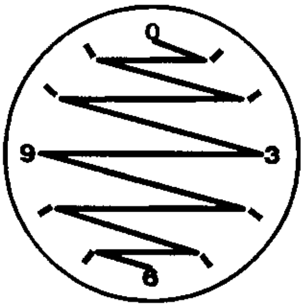
The notes begin with A natural, and then alternately rise and fall, in the sequence B flat, G sharp, B natural, G natural, C natural, F sharp, C sharp, F natural, D natural, E natural, and D sharp. I find it convenient to number these notes according to their signed distances from A natural, which I number as 0. The twelve notes are then seen as

```
0 1 _1 2 _2 3 _3 4 _4 5 _5 6
```

And it simplifies things if we take these mod 12, giving

```
0 1 11 2 10 3 9 4 8 5 7 6          [A]
```

I have found it helpful visually to write these numbers as the hours on a clock face (using 0 in place of 12), and to connect the hours by lines in the order given, that is, draw a line connecting 0 to 1, 1 to 11, 11 to 2, and so on, ending with a line drawn from 7 to 6.



This clock figure makes more apparent various symmetries that reduce the number of permutations that need to be considered.

If we take the first difference of [A], we get the following:

```
1 10 _9 8 _7 6 _5 4 _3 2 _1
```

and if we take this mod 12, we get

```
1 10 3 8 5 6 7 4 9 2 11          [B]
```

and it is easy to see that the list [A] is a 0-origin permutation having a first difference, mod 12 [B] which is a 1-origin permutation. Thus we have transformed the musical problem, having to do with twelve-tone rows, into the combinatorial problem of determining all the permutations of `i. 12` having a first difference which is a permutation of `>: i. 11`. That is, we want to know how many such permutations there are, and what they are. To make it easier to discuss "a permutation having a first difference mod permutation length also a permutation". I'll call such an object a *'dil'* (from *Distinct Interval List*).

There are 479,001,600 permutations of `i. 12`, so it is a large problem to sift through these permutations looking for dils. For example, to load the table of all permutations of order 12 would take 4*12*!12, or 22,992,076,800 bytes. I believe that this would be impossible to load in real memory on the largest contemporary machine. This paper explores ways to cut it down to a more manageable size.

I heard the problem in the early 1960s when Iverson notation was available only on the printed page, and worked at it by hand for several months without making much progress. Recently I decided to tackle it once more, beginning by studying the permutations of smaller order. I found that dils occur only among even length permutations. The order two permutations are easy: both are dils: 0 1 and 1 0, having an interval of 1. These can be done mentally, but it quickly becomes necessary to develop programming tools to aid in the exploration:

```
   pt=.i.@! A. i.      NB. permutation table
   mfd=.# | }. - }:    NB. modular first difference
   mn=. -: ~.          NB. distinct items?
   dil=.mn@mfd"1       NB. a dil?
   dils=. dil # ]      NB. all dils
   pt 3
0 1 2
0 2 1
1 0 2
1 2 0
2 0 1
2 1 0
   mfd 0 1 5 2 4 3
1 4 3 2 5
   mn mfd 0 1 5 2 4 3
1
   dil 0 1 5 2 4 3
1
```

Studying the dils of order 4 give us some insight into the problem:

```
   dils pt 4      NB. dils of length 4
0 1 3 2
0 3 1 2
1 0 2 3
1 2 0 3
2 1 3 0
2 3 1 0
3 0 2 1
3 2 0 1
```

Some symmetries are present that will let us cut the problem down in size. Only permutations beginning with 0 need be considered, since the others can be obtained by clock face rotations:

```
   ro=. #@] | +   NB. rotate y by x
   1 ro 0 1 3 2
1 2 0 3
   2 ro 0 1 3 2
2 3 1 0
```

and similarly for the others. I call the dils beginning with zeros 'basic dils', since all the others can be obtained from them by rotation, or, in musical terms, by transposing. By looking for dils only among permutations beginning with 0, our order !12 problem has been reduced reduced to an order !11 problem, or 39,916,800. Here are the basic dils of orders 4, 6, and 8:

```
   a4=.dils(i.!3)A. i.4
   a6=.dils(i.!5)A. i.6
   a8=.dils(i.!7)A. i.8
   a4              a6                    a8
0 1 3 2         0 1 5 2 4 3         0 1 3 6 2 7 5 4
0 3 1 2         0 2 1 4 5 3         0 1 6 5 3 7 2 4
                0 4 5 2 1 3         0 1 7 2 6 3 5 4
                0 5 1 4 2 3         0 1 7 3 6 5 2 4
                                    0 2 1 5 3 6 7 4
                                    0 2 3 6 5 1 7 4
                                    0 2 5 1 7 6 3 4
                                    0 2 7 6 1 5 3 4
                                    0 3 1 2 6 5 7 4
                                    0 3 2 7 1 5 6 4
                                    0 3 5 1 2 7 6 4
                                    0 3 5 6 2 1 7 4
                                    0 5 3 2 6 7 1 4
                                    0 5 3 7 6 1 2 4
                                    0 5 6 1 7 3 2 4
                                    0 5 7 6 2 3 1 4
                                    0 6 1 2 7 3 5 4
                                    0 6 3 7 1 2 5 4
                                    0 6 5 2 3 7 1 4
                                    0 6 7 3 5 2 1 4
                                    0 7 1 5 2 3 6 4
                                    0 7 1 6 2 5 3 4
                                    0 7 2 3 5 1 6 4
                                    0 7 5 2 6 1 3 4
```

Further efficiencies are possible. Notice that all of these dils not only begin with the constant 0, but end with a constant that is half of the order: 2, 3, and 4 for orders 4, 6, and 8, respectively. This means that in searching for dils we only have to look at those permutations beginning with 0 and ending with a constant, with some permutation between them.

The desired inner permutation is given by:

```
   si=. i. -. 0: , -:   NB. integers thro n-1, less 0 and -:n
   si 2
   si 4
1 3
   si 6
1 2 4 5
   si 8
1 2 3 5 6 7
   si 10
1 2 3 4 6 7 8 9
   si 12
1 2 3 4 5 7 8 9 10 11
```

By having to consider only inner permutations of order n-2, we have now reduced our problem to one of order !10, or 3,628,800. Furthermore, looking carefully again at the tables a4, a6, and a8 above, we see that only the first half of the basic dils need to be tested, since the rest can be found by clock face reflections in the y-axis. That is, any one of the rows in the lower half of any of these tables is obtainable from one of the rows in the upper half. The verb `ry` reflects a dil about the y-axis:

```
   ry=. # | # - ]
   ry 0 1 3 2
0 3 1 2
```

This means that to find the dils of order 12, we have to test only `-:!10`, or 1,814,400 permutations. This is a reduction from !12 by a factor of 264.

Since we can always retrieve a dil if we know its atomic number and its length, we don't need to exhibit the complete row. It suffices to obtain only its atomic number. For example, the dils of order 4 can be obtained using only 8 integers, rather than the 32 required by the display of the four atoms of each permutation form of the dil. We can define a verb dan to give us the dils in atomic number form:

```
   dan=. (dil # A.)    NB. dil atomic number
   dan pt 4            NB. atomic numbers of dils of order 4
1 4 6 8 15 17 19 22
```

There are two additional clock face reflective symmetries in these dils. In addition to the y-axis symmetry mentioned above, there are reflections possible in the x-axis, and in both the x and y axes. For example, the dil:

```
   r=. 0 1 3 2 7 10 8 4 11 5 9 6
```

can be reflected in the x-axis by:

```
   rx=. [: |. # | -:@# - ]
   rx r
0 9 1 7 2 10 8 11 4 3 5 6
```

and in the x-y axes by:

```
   rxy=.[: |. # | -:@# + ]
   rxy r
0 3 11 5 10 2 4 1 8 9 7 6
```

I haven't found a way to use these further symmetries to reduce the work necessary to solve the dil problem. The program I use to find the primitive dils of order n is:

```
   pdon=. 3 : 0
NB. argument is 4-item list, e.g. pdon 12 5040 0 1814400
 'nibm'=.y.
NB. n is length of permutation
NB. i is size of batch (depends on memory size and n)
NB. b is base index (usually 0 initially)
NB. m is maximum item number (usually -:!n-2)
NB. z is result, list of indices of primitive dils of order n
 z=.''
 s=.si n            NB. for example, si 8 is 1 2 3 5 6 7
 h=.-:n             NB. for n=8, h is 4
 while. b<m do.
   t=.0,.((b+i.i)A. s),.h  NB. provide another batch
   z=.z,dan t          NB. append primitive dil atomic #s to z
   b=.b+x              NB. step base by batch size
 end.
z
)
```

The line assigning t shows the utility of being able to specify the right argument to the `A.` primitive. On my computer, it took about 10 minutes to compute the dils of order 10. I don't know how long it took to do those of order 12. I started it going just before I went to bed, and it was ready in the morning.

For the record, the number of dils of orders 2 through 12 are:

order	primitive dils	basic dils	all dils
2	1	1	2
4	1	2	8
6	2	4	24
8	12	24	192
10	144	288	2880
12	1928	3856	46272

Here are a few nicely symmetrical dils of order 12:

```
    pty12s=.646517 3154657 4275293 5762095 7289175 9306655
    pty12s=. pty12s, 11633649 12187013 13754599 14826363
 16823821

    pty12s A. i.12
  0 1  3 10  2  5 11 8  4  9  7 6
  0 1 10  8  3 11  5 9  2  4  7 6
  0 2  3 10  1  5 11 7  4  9  8 6
  0 2  7 10 11  3  9 5  4  1  8 6
  0 3  1  2 10  5 11 4  8  7  9 6
  0 3  7  8 10  5 11 4  2  1  9 6
  0 4  3  1  8  5 11 2  7  9 10 6
  0 4  5  8  3  1  7 9  2 11 10 6
  0 4  9 11  2  1  7 8  5  3 10 6
  0 5  1 10  8  9  3 2  4  7 11 6
  0 5  8  4  3  1  7 9 10  2 11 6
```

If you're a musician you might try playing these. They also make interesting clock face patterns. If you have a current version of J on your computer you can see them drawn using the graphics facilities available. The functions `sogwin` and `sline` are available if you have profile.js in the command line as advised in installing the system. Additional information about using the J graphics facilities are described in the book *'Fractals Visualization and J'* by Clifford Reiter, available from Iverson Software, Inc.

Here is the beginning of a sample session of visualizing dils on a clock face to help you get started:

```
    ]r12=: 12 %: _1            NB. 12th root of negative 1.
  0.965926j0.258819
    all=. r12^2*i.12           NB. first 12 powers of this root
    ]coords=. +.all            NB. real & imaginary parts
           1           0
    0.866025         0.5
         0.5    0.866025
  6.12574e_17          1
        _0.5    0.866025
   _0.866025         0.5
          _1 1.22515e_16
   _0.866025        _0.5
        _0.5   _0.866025
 _1.83772e_16         _1
         0.5   _0.866025
    0.866025        _0.5

    scaled=. 500*1+coords   NB. scale to screen coordinates
```

```
   1000      500
933.013      750
    750  933.013
    500     1000
   250  933.013
66.9873      750
      0      500
66.9873      250
    250  66.9873
    500        0
    750  66.9873
933.013      250
```

With these defined you can create a graphics window with:

```
   sogwin 'scaled'
   0 sline scaled
```

And display the lines for a given permutation on the clock face with

```
   perm=. 12 | 3+ry 0 1 11 2 10 3 9 4 8 5 7 6
   p=. perm{scaled
   0 sline p
```

The definitions of some of the graphics verbs needed are given below:

```
   sogwin =. 3 : 0
3 3 500 500 sogwin y.  :
x=.<.x.%2.5
z=.'pc ',y.,';xywh ',(": x),';cc g isigraph;pas ',":2{.x
wd z,';pcenter;pscale;pcloseok;pshow sw_showna;'
)

  sline =. 3 : 0"1 2
0 0 0 sline y.
:
wd 'grgb ',(":x.),'; gpen 1 ps_solid;'
wd 'gmove ',(":{.y.),';'
wd z=:,'gline ',"1 (":}.y.),"1 ';'
wd 'gshow;'
)

    spoly =. 3 : 0"1 2
wd 'gpolygon ',(,' ',.":y.),';gshow;'
:
sfill x.
spoly y.
)
```

Elegant Programming

by Chris Burke

Part 1 &. dfh

J defines inverses for many functions, and provides various ways of making use of them. A recent addition to the set of inverses in J2.06, namely the inverse to `n&#.`, enables the elegant title expression, and prompted this note. We will look first at the title expression, and then examine how it works.

First define:

```
    dfh=. 16&#. @ ('0123456789ABCDEF'&i.) NB. decimal from hex

    hex=. &. dfh
```

Then:

```
    'FEED' + hex 'B'
FEF8

    'FF' * hex '101'
FFFF
```

Thus, `hex` is an adverb which returns a verb that works in hexadecimal.

Under

The definition of `hex` uses the conjunction `&.` (*under*). Given verbs u and v where the inverse v^{-1} is defined, then `u&.v` is equivalent to v^{-1} u v.

Here are some examples:

inverse of natural log is the exponential:

```
    3 + &. ^. 4
12
```

inverse of reciprocal is itself:

```
    am=. +/ % #                    NB. arithmetic mean
    hm=. am &. %                   NB. harmonic mean
    hm 2 3 5
2.90323
```

inverse of open is box:

```
      n=. 'winston';(i.3 4);10 20
      # &.> n
┌─┬─┬─┐
│7│3│2│
└─┴─┴─┘
      $ &.> n
┌─┬───┬─┐
│7│3 4│2│
└─┴───┴─┘

      each=. &.>
      |. each n
┌───────┬──────────┬─────┐
│notsniw│8 9 10 11 │20 10│
│       │4 5  6  7 │     │
│       │0 1  2  3 │     │
└───────┴──────────┴─────┘
```

inverse of the binary representation is the base-2 value:

```
      bitwise=. &.#:

      5 +. bitwise 6                      NB. bitwise OR
7

      5 *. bitwise 6                      NB. bitwise AND
4

      5 ~: bitwise 6                      NB. bitwise XOR
3
```

inverse of transpose is itself:

```
      +/\ i.3 4                           NB. accumulate along columns
 0  1  2  3
 4  6  8 10
12 15 18 21

      +/\ &.|:  i.3 4                     NB. accumulate along rows
0  1  3  6
4  9 15 22
8 17 27 38
```

Inverse

You can access the inverse directly using `^:_1` (power of minus 1). For example, define:

```
   inv=. ^:_1
```

inverse of add 2 is subtract 2:

```
   +&2 inv
┌─┬─┬─┐
│-│&│2│
└─┴─┴─┘

   +&2 inv 1 2 3
_1 0 1
```

inverse of sum scan is first differences:

```
   +/\ inv 2 3 5 7 11
2 1 2 2 4
```

inverse of product scan is rate of increase:

```
   */\ inv 2 3 5 7 11
2 1.5 1.66667 1.4 1.57143

   ({.,}.%}:) 2 3 5 7 11
2 1.5 1.66667 1.4 1.57143
```

inverse of `p:` *determines number of smaller primes:*

```
   p: 100000                  NB. 100000'th prime
1299721

   p: inv 1299721             NB. number of primes less than 1299721
100000
```

Obverse

You can define inverses for use with the conjunctions `&.` and `^:` directly, using the conjunction `:.` (obverse). The result of `u :. v` is a verb that is equivalent to u with an assigned obverse `v`.

In general, the term *obverse* is used instead of *inverse,* since the defined obverse need not be a true inverse, indeed it may be an unrelated verb.

For example:

```
   %: (*:2) + (*:5)           NB. square root of 2² + 5²
5.38516
```

```
    2 + &. *: 5                       NB. same
5.38516

    2  +&.(*: :. (^&1r2)) 5           NB. same (inverse of *: is square root)
5.38516

    2  +&.(*: :. (^&1r3)) 5           NB. using cube root as obverse
3.07232
```

Hex

Now let's take a closer look at the definition of `hex`:

```
    dfh=. 16&#. @ ('0123456789ABCDEF'&i.)

    hex=. &. dfh
```

The inverse of `'0123456789ABCDEF'&i.` is:

```
    { & '0123456789ABCDEF'
```

while the inverse of `16&#.` is:

```
    16 16 ... 16 &#:
```

with as many 16's as required.

Also, the inverse of `f @ g` is g^{-1} `@` f^{-1}, hence the inverse of `dfh` can be calculated.

For any verb `f`:

```
    f hex x   <=>      f &. dfh x        <=>       dfh⁻¹ f
dfh x
```

Part 2 #~ 1: = #@q:

This elegant expression uses the verb `q:` introduced in J2.06, and provides an interesting exercise for the newcomer to J. The expression contains a hook, a fork, an adverb, a conjunction, and a constant function. It is fair to say that once you understand this expression, then you also understand the essence of functional programming in J.

First let's use the definition, as follows:

```
      p=. #~ 1: = #@q:

      p 2+i.30
2 3 5 7 11 13 17 19 23 29 31

      p 123456+i.50
123457 123479 123491 123493 123499 123503
```

Thus, p selects the primes in a list of positive integers. How does it work?

The definition of p can be read from left to right as: *select where 1 is the number of prime factors.*

Let's build up this definition step by step.

`q:` returns the prime factors of its argument, for example:

```
      q: 123456
2 2 2 2 2 2 3 643
```

The verb `#@q:` returns the count of the number of prime factors. Here, the conjunction `@` (atop) creates a new verb that applies `#` to the result of `q:`.

```
      p0=. #@q:

      p0 123456
8
```

We now want to generate a boolean where a 1 indicates a prime, i.e. where the count of the number of prime factors is 1. This is achieved by the following fork:

```
      p1=. 1: = p0

      p1 2+i.12
1 1 0 1 0 1 0 0 0 1 0 1
```

The `1:` needs some explanation. A fork is a sequence of three verbs, `f g h` where:

```
      (f g h) x        <=>        (f x) g (h x)
```

Note that the three elements of a fork are *verbs*. In the definition of `p1`, the leftmost verb is `1:`, which is a verb that returns the value 1, given any argument. Here, J makes clear the difference between a number, and a verb that returns that number. This distinction is not made in standard mathematical notation, which can be confusing.

We now use the boolean to select the primes. The selection verb is `#`, which takes a boolean left argument:

```
   1 1 0 1 0 1 # 2 3 4 5 6 7
2 3 5 7
```

In this case, however, we want to use `#` with the boolean as a right argument. To do so, we create a new verb with the adverb `~` (passive), that swaps its arguments:

```
   p2=. #~

   2 3 4 5 6 7 p2 1 1 0 1 0 1
2 3 5 7
```

(You can read `#` as *select*, and `#~` as *select where*.)

Finally, we define `p` as the hook `p2 p1`, giving us the original definition of `p`. A hook is a sequence of two verbs `f g`, where:

```
      (f g)x        <=>        x f g x
```

Box Display

Box display helps clarify the structure of functional expressions, and should be used by default as a learning aid. You graduate from the school of functional programming when you find that you no longer need box display to read functional expressions! For example:

```
   p
┌─────┬───────────────┐
│┌─┬─┐│┌──┬─┬────────┐│
││#│~│││1:│=│┌─┬─┬──┐││
│└─┴─┘││  │ ││#│@│q:│││
│     ││  │ │└─┴─┴──┘││
│     │└──┴─┴────────┘│
└─────┴───────────────┘
```

In box display, elements are grouped into twos and threes, as follows:

2 elements: if the right element is an adverb, then the two elements represent the verb formed by applying the adverb to the argument on its left, otherwise, the two elements are a train (in the case of 2 verbs, a hook).

3 elements: if the center element is a conjunction, then the three elements represent the verb formed by applying the conjunction to its two arguments, otherwise, the three elements are a train (in the case of 3 verbs, a fork).

With this in mind, we can see from the above box display that p is a hook. The left element is the result of applying the adverb ~ to the verb #. The right element is a fork, whose rightmost element is the result of applying the conjunction @ to the verbs # and q:.

Here are a couple more examples to illustrate:

```
     t0=. +/ @ (1: = (+. i.))

     t0
┌─────┬─┬──────────────┐
│┌─┬─┐│@│┌──┬─┬───────┐│
││+│/││ ││1:│=│┌──┬──┐││
│└─┴─┘│ ││  │ ││+.│i.│││
│     │ ││  │ │└──┴──┘││
│     │ │└──┴─┴───────┘│
└─────┴─┴──────────────┘
```

Here t0 is the result of applying the conjunction @, with a left argument of sum (+/), and a right argument of a fork, whose rightmost element is the hook (+. i.).

The expression +. i. computes the GCD (+.) of an integer and all the integers below it. For example,

```
     (+. i.) 12
12 1 2 3 4 1 6 1 4 3 2 1
```

Thus, t1 defined below takes an integer argument and computes which integers below it are relatively prime to it (i.e. the GCD is 1):

```
     t1=. 1: = (+. i.)

     t1 12
0 1 0 0 0 1 0 1 0 0 0 1
```

Therefore `t0`, which is defined as `+/ @ t1`, is Euler's totient function, i.e. the number of integers below a number that are relatively prime to it.

```
      t0 12
4
```

A more efficient version is:

```
      t3=. * -.@%@~.&.q:

      t3 12
4
```

Can you read its box display?

```
      t3
+-+-----------------------+
|*|+---------------+--+--+|
| ||+--------+-+--+|&.|q:||
| |||+--+-+-+|@|~.||  |  ||
| |||-.|@|%|| |  ||  |  ||
| |||+--+-+-+| |  ||  |  ||
| ||+--------+-+--+|  |  ||
| |+---------------+--+--+|
+-+-----------------------+
```

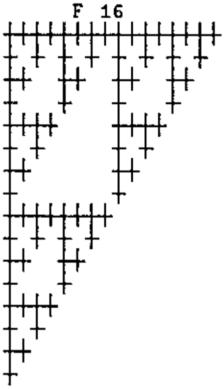
A Fractal Verb in J

by Richard Oates

At Christmas I sent a copy of J to a professor at Cornell College in Iowa where I went to school. I also sent this verb. It prints one line at a time like Norman Thomson's program in Vector Vol.11 No.3 page 17.

Primitives like + and programs like `F` are verbs. `F` makes a square fractal.

```
      F 16
```



Consider the sentence "She walks fast.". The adverb "fast" changes the meaning of "walk". English doesn't need a "walkfast" verb. Consider the composition (+/). The Insert adverb (/) sticks the Plus verb (+) between each item of the argument. J doesn't need a `SUM` built-in function.

```
      +/ 1 2 3   NB. +/ changes 1 2 3 to 1+2+3
  6
```

J programs can be composed of nothing but verbs. You don't have to refer to application data. Iverson calls this "tacit definition". `F` is a tacit program. The execution sequence is mapped by `Track`. In a track map each composed subverb is pegged with a horizontal line.

```
F=. Kill@Draw@({.&1)
Kill=. (i.0 0)"_
Draw=. Next@NEIP^:#
Next=. [ Screen
Screen=. {&' ┼' (1!:2) 2:
NEIP=. ~:/\
```

```
      Track'F Kill Draw'
F                          Kill    Draw
│──────────── @ ──────│  │□ "-│  │──────────── ^:#│
│Kill@Draw      {.&1  │  │    │  │Next@NEIP       │
└─────────────────────┘  └────┘  └────────────────┘

      Track'Next Screen NEIP'
Next        Screen            NEIP
│[ Screen│  │──── ──── 2:│  │─── \│
└────────┘  │{& ── 1!:2  │  │~:/  │
            │    +       │  └─────┘
            └────────────┘
```

A verb applies to a noun on each side (like +) or to a noun on the right (like `F`). An adverb applies to an argument on the left (like Insert). The argument may be verb or noun. A conjunction applies to an argument on each side. The result of an adverb or conjunction is a verb. The result of a verb is a noun. Verbs, adverbs, conjunctions and nouns can all be specified with the copula (`=.`).

Atop (`@`) and Bond (`&`) are conjunctions. In `F`, Atop runs the (`Kill@Draw`) verb after or "atop" the (`{.&1`) verb. The latter is composed of Take (`{.`), Bond, and 1. Names of primitives are constructed from one or two characters. If two, the second is period (`{.`) or colon. This is what (`{.&1`) does.

```
     4 {. 1 5 0 3 2 3    NB. No composition, two arguments for {.
1 5 0 3
     4 {. 1              NB. J supplies zeroes if necessary
1 0 0 0
     {.&1 (4)            NB. Composed verb ({.&1) applied to 4
1 0 0 0
```

The argument of `F` is the argument of (`{.&1`). If the argument is `4` the result of (`{.&1`) is (`1 0 0 0`) and (`1 0 0 0`) is the argument of (`Kill@Draw`). In `Draw` the Power (`^:`) conjunction runs (`Next@NEIP`) `#` (`Tally`) times. The argument of `#` is (`1 0 0 0`) as is the initial argument of (`Next@NEIP`).

```
    # 1 0 0 0
 4
```

`NEIP` (`~:/\`) makes the fractal. Not Equal (`~:`) is a verb. Insert (`/`) and Prefix (`\`) are adverbs. `NEIP` applies Not Equal Insert (`~:/`) to each prefix. Prefix structures its argument as a bunch of prefixes: the first item, the first two, the first three, etc.

```
      <\ 1 0 0 0          NB. Box (<) each prefix (\)
┌─┬───┬─────┬───────┐
│1│1 0│1 0 0│1 0 0 0│
└─┴───┴─────┴───────┘
```

This is what J does when Not Equal Insert is applied to the last prefix (`1 0 0 0`) in place of Box.

insert verb	`1 ~: 0 ~: 0 ~: 0`
group verbs right	`1 ~: (0 ~: (0 ~: 0))`
`0~:0` is false	`1 ~: (0 ~:      0    )`
`0~:0` is false	`1 ~:       0`
`1~:0` is true	`1`

The result is also `1` when Not Equal Insert is applied to any other prefix. The result of `NEIP` on (`1 0 0 0`) is (`1 1 1 1`). In `Next`, `Screen` displays '┼┼┼┼' for (`1 1 1 1`), but Left (`[`) returns (`1 1 1 1`) when applied to (`1 1 1 1`) and '┼┼┼┼'.

```
   NEIP 1 0 0 0
1 1 1 1
   Next@NEIP 1 0 0 0
┼┼┼┼
1 1 1 1

   NEIP NEIP 1 0 0 0
1 0 1 0
   Next@NEIP Next@NEIP 1 0 0 0
┼┼┼┼
┼ ┼
1 0 1 0
   Next@NEIP ^: # 1 0 0 0
┼┼┼┼
┼ ┼
┼┼
┼
1 0 0 0
   F 4
┼┼┼┼
┼ ┼
┼┼
┼
```

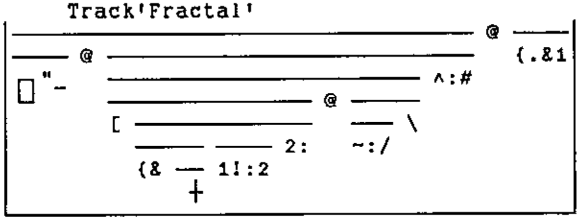
A scalar verb like Minus (`-`) is applied to each item of a list. An underbar (`_`) touching a number is a negative sign, not a verb. Scalar technique is extended to all verbs with the Rank (`"`) conjunction. Rank applies a non-scalar verb to each part of its argument.

```
   1 - 6 1.2 4
_5 _0.2 _3
   - 6 1.2 4
_6 _1.2 _4
   F"0 (2 0 4)
┼┼
┼

┼┼┼┼
┼ ┼
┼┼
┼
```

Tacit definition sharpens thought. Also, it facilitates the kind of pre-chip manipulation that is increasingly required for parallel and non-parallel hardware. The Fix (`f.`) adverb substitutes definitions for names. When `Track` is applied to a single name the linear definition appears below the map.

```
      Fractal=. F f.
      Track'Fractal'
```



```
(i.0 0)"_@(([ ({&' ┼' (1!:2) 2:))@(~:/\)^:#)@({.&1)
```

The right argument of many conjunctions is executed before the left. In `Fractal`, (`[` ——) is `Next` and (—— —— `2:`) is `Screen`. Two isolated verbs are a "hook" and three are a "fork". A fork applies the first and last verbs to its argument, and then applies the middle verb to their results. A hook applies the last verb to its argument, and then applies the first verb to its argument and to the result of the last verb.

Vocabulary

Verbs	
`+`	Plus
`{.`	Take
`i.`	Index of
`#`	Tally
`[`	Left
`{`	From
`~:`	Not Equal
`2:`	Two
`-`	Minus
`<`	Box
`F`	Map fractal
`Kill`	Display nothing
`Draw`	Main
`Next`	Skip '┼' and blank
`Screen`	Display as '┼' and blank
`NEIP`	Not Equal Insert Prefix
`Fractal`	Map fractal
`Track`	Simplify boxed representation

Adverbs	
`/`	Insert
`\`	Prefix
`f.`	Fix

Conjunctions	
`@`	Atop
`&`	Bond
`"`	Rank
`"`	Constant
`^:`	Power
`!:`	Foreign

Nouns	
`1`	One
`0 0`	Zero zero
`_`	Infinity
`2`	Two

Two English words like *A* and *Ad* usually have nothing in common. Two J words like `{` and `{.` frequently do. From selects any. Take selects from the top or bottom. Rank is (verb"noun). Constant is (noun"noun). Foreign links the operating system. Write is `1!:2`. Boxed representation is `5!:2`. Underbar (`_`) is infinity only when isolated.

Tilting at Windmills: a New Attack on Nested Arrays

by Douglas R. Bohrer

Abstract

This article is a proposal for a new notation for nested arrays, a notation the author feels is more consistent with the simple array APL concepts used before nested arrays were implemented. The proposal is to have all simple array APL functions work at the simple array level just as scalar functions work at the scalar level. An operator would be used to make simple array functions into nested array functions which would work on the simple array elements of nested arrays in ways similar to the way the simple array functions worked on the scalar elements of simple arrays. Motivations for the proposal are discussed.

Simple Arrays

Back in the days before nested arrays, there were only objects which I will refer to as *simple arrays*. The elements of simple arrays were scalars, usually of uniform data type. Scalar functions were defined for arrays as working element-by-element through the arrays, that is at the lowest level of the array. *Scalar expansion* was defined by saying that the scalar argument would be used for each of the scalar elements in the array. The functions which manipulate simple arrays, like *rho* or *ravel* or *iota*, I will call **simple array functions**. Simple array functions are defined to include operator-derived functions like inner and outer product.

Nested Arrays

I will define **nested arrays** as arrays in which each element is a simple array or scalar. Just as scalar functions work at the scalar level, I define simple array functions to work at the simple array level. This means that simple array functions work element-by-element through nested array arguments. Below, I will refer to this definition as *Rule 1*.

A simple array argument could be expanded by saying that the simple array argument would be used for each of the simple array elements in the nested array. Below, I will refer to this definition as *Rule 2*.

I need to expand the definition of nested arrays by saying that nested arrays may have elements that are themselves nested arrays. Rule 1, simple array functions, and Rule 2, simple array expansion, would then be applied recursively until a simple array was reached.

For ease in illustrating, I will display nested arrays as enclosed in braces, with simple arrays separated by semi-colons. For example, if *A*,*B*,*C* and *D* are simple arrays, then `{A;B} + {C;D}` is defined as `{A+C;B+D}` according to Rule 1. Similarly, Rule 1 indicates that `+/{A;B;C}` is defined as `{+/A;+/B;+/C}` because `+/` is a simple array function. Simple array expansion means that `A,[1]{B;C}` is defined as `{ A,[1]B ; A,[1]C }` according to Rule 2.

To illustrate the recursive use of Rules 1 and 2 I will use `{A;{B;C}}` to indicate a nested array in which the first element is the simple array *A*, and the second element is a nested array of simple arrays *B* and *C*. Then if *D* and *E* are simple arrays `{A;{B;C}}-D` is defined as `{A-D;{B-D;C-D}}` by applying Rule 2 recursively. Similarly, `{A;{B;C}}-{D;E}` is defined as `{A-D;{B-E;C-E}}` using both rules.

Nested Array Functions

Nested array functions will be derived from simple array functions with a left operator, the dollar sign. I think `$` is a good choice because it is a widely available character that currently isn't being used for anything by APL. A derived nested array function is defined as working on nested arrays in a way similar to the way the simple array function works on simple arrays. For example:

> `{A;B} $, {C;D}` is defined as `{A;B;C;D}`
> just as `1 2, 3 4` is `1 2 3 4`.
>
> A little bit more complex,
>
> `{A;B} $+.= {C;D}` resolves to `(A=C)+(B=D)`
> just as `1 2+.=3 4` is `(1=3)+(2=4)`.
>
> It should be noted that `$ρ {A;B;C}` is defined as 3 but
> `ρ{A;B;C}` is defined as `{ρA;ρB;ρC}` which is not the same.

The definition of `$` above allows making a nested array containing two nested array elements, that is of depth greater than 2 using enclosure, that is

> `(⊂{A;B}) $, ⊂{C;D}` defined as `{{A;B};{C;D}}`.

A New Null Element

One of the implications of this scheme is that there will be a null nested array with a nested length of 0. The notation for this would be defined so that

> `{A;B}  $,  0  $ρ  X` gives `{A;B}` as a result where *X* is any scalar, simple array or nested array.

The depth of the null nested array is 2. It's not clear what the simple array rho of the null nested array should be. Perhaps zero would be convenient.

Simple Array Indexing

It may be redundant but I feel compelled to start my discussion of indexing by reminding the reader that `{A;B}[C]` is defined as `{A[C];B[C]}` because indexing is a simple array function which according to Rule 1 applies element by element to a nested array. For a nested array index to a simple array Rule 1 implies that `A[{B;C}]` is defined as `{A[C];B[D]}`. I think that similar methods can easily be used to resolve simple indexing in higher dimension objects for nested arrays. I will leave this elaboration to the student as an exercise. (I always wanted to say that.) It then seems logical to look at simple indexed assignment in similar fashion:

> `{A;B}[C]  D` is defined as `{A[C]  D;B[C]  D}` using both Rule 1, the indexed assignment applying to the elements of the nested array, and Rule 2, the simple array being expanded.

> For the more complex case `{A;B}[C]  {D;E}` is defined as `{A[C]  D;B[C]  E}` where *A*, *B*, *C*, *D* and *E* are simple arrays.

> For the trickiest case, if *F* is also a simple array then `{A;B}[{C;D}],{E;F}` is defined as `{A[C],E;B[D],F}` using Rule 1 element-by-element for the nested array index as well as the other nested array arguments.

Nested Array Indexing

Nested array indexing works on nested arrays in a fashion similar to the way simple indexing works on simple arrays. Indexing a nested array with a simple array index gives nested array results whose simple array elements are positioned like the scalar elements would be for a simple array indexed by a simple array. For example, `{A;B;C;D}$[1 3 1 4]` is defined as `{A;C;A;D}` just as `'ABCD'[1 3 1 4]` would be `'ACAD'`. In a special case, the result of a nested array indexed by a scalar is a simple array just as a simple array indexed by a scalar is a scalar. For example, {*A*;*B*;*C*}$[1] is defined as *A* a simple array result, just as the result of '*ABC*'[1] is a scalar '*A*'.

For a nested array index to a nested array, the most useful definition is not obvious. I think that the best thing to do is to define it as a nested array of the results from each simple array element of the index applied to the nested array. This definition would mean, for example, that {*A*;*B*;*C*;*D*}$[{{1 3};{2 4}}] would be {{*A*;*C*};*B*;*D*}}.

For the multi-dimension case, the result would be a nested array which exhausted every combination of simple array elements. If we define, for example, *NN* as a nested matrix then *NN*$[{*A*;*B*} ; {*C*;*D*}] would be 2 2$⍴{*NN*$[*A*;*C*];*NN*$[*A*;*D*];*NN*$[*B*;*C*];*NN*$[*B*;*D*]} where each of the indexing operations on *NN* would follow the method for simple array indexes of nested arrays. This definition is consistent with the case where if *A* is a simple matrix then *A*[1 2;3 4] is 2 2⍴*A*[1;3],*A*[1;4],*A*[2;3],*A*[2;4].

User Functions

Should user-defined functions be simple array functions or nested array functions? Probably the user should have the option to define them either way. If a dollar sign appeared in the header just before the function name, the function would always be a nested array function. If called with nested array arguments, such a function would be passed the arguments as nested arrays.

The default should be that all user functions would be assumed simple array functions. Such functions would use Rule 1 to process nested arguments. If *UFUN* is a user-defined function without a $ in the header, then *UFUN* {*A*;*B*} would be {*UFUN A*;*UFUN B*} and *UFUN* itself would not be able to see that it had been called with a nested argument.

The nesting operator should work for user functions as well as native interpreter functions. The operator $ placed before a user function name would feed the nested array argument into the function instead of calling it repeatedly using Rule 1. It would then be possible to define functions that could work either as simple array functions or nested array functions.

Why Bother?

I think this scheme is "more APL-like" than current nested array implementations and would therefore be less confusing and easier to learn. Let me expand on this point.

I have felt for some time that the shift from simple array APL to nested array APL is needlessly confusing. I think the confusion comes from the difference between the behaviour of scalar functions on arrays and the need to disclose nested array elements to work on them. If scalar functions worked on simple arrays the way most functions work on nested arrays, then you would write *A*+¨*B* to add all the elements of simple arrays *A* and *B* together. Unconsciously, the student expects that the "for each" will be assumed for nested arrays just as it is for scalar functions working on simple arrays.

This confusion is NOT a figment of my imagination nor the result of a unique personal learning difficulty. Since I first wrote about this in 1982, I have had the opportunity to talk to lots of APLers about nested arrays. Most thought learning them difficult. Some found them so confusing they don't use them at all. Even several APL fanatics at the APL91 conference admitted to me that they don't use nested arrays because they're too confusing to be worth the trouble.

The scheme I propose has a lot of educational economy built into it. It builds on the methods of handling simple arrays to introduce nested arrays. You already know how nested array index or nested array catenate is going to work because it works just like the simple array function does. A similar method was used to add networking commands to UNIX. The UNIX remote copy command, `rcp`, works just like the ordinary copy command, `cp`, except that it works using somebody else's machine. Think of the leading "`r`" as an operator.

In contrast with the scheme I propose, current methods have *negative* educational economy. All current methods of nested array implementation have formerly predictable simple array functions doing seemingly arbitrary things with nested arrays. These behaviours when applied to unintentional nested arrays created by strand notation can get even experienced APL programmers into deep debugging doo-doo. Here I speak from personal experience of working with 5- to 20-person APL teams for over 5 years. As the "debugger of last resort" I tracked down a lot of these problems for everyone from the beginners to APLers who had years' more practical experience than I did.

I think this state of affairs is unfortunate because it is limiting our ability to teach APL to the masses. It used to be that APL required the student to learn only as much as was required for the problem at hand. What he didn't know couldn't hurt him because he wouldn't use it accidentally. The current implementations of nested arrays with strand notation make what the student doesn't know dangerous, causing problems with objects he will be unable to identify, let alone fix on his own.

Where Do We Stand Now?

I have great confidence that this proposal has merit. I first wrote about it in March of 1982. While I have refined the idea a lot since then, I have not changed the basic scheme much at all.

I do not have any confidence at all that this proposal will receive any serious consideration from implementers, the APL Standards Committee, or anybody influential or powerful. The reactions of all of these people when I discussed this with them have not changed in the last 13 years. It is uniformly assumed that (a) I have an obvious learning disability because I can't learn to love nested arrays as they are; (b) I am extremely silly to suspect that people of superior intellect, such as those who designed nested array implementations, could be as fallible as any ordinary mortal, such as myself; and (c) I am naive in the extreme to expect that the greedy capitalists who pay the bills for implementers would ever write off the investment made in implementing the current design.

Standards Deadlock Broken

At APL91, the Standards Committee representatives jubilantly reported that they had reached agreement on a standard for nested arrays. All previous disagreement had been resolved miraculously. How did this miracle happen? Did the Lord come to all the members in a dream and tell them what the standard should be? Unfortunately, the result seems to have been far closer to the "Barebones" Parliament of Cromwellian England. It seems that all of the representatives who had argued in past deliberations against the position the committee approved were not present at the meeting where the proposal was adopted. Their companies were no longer interested, having left the APL language field.

I'm not convinced that exhaustion necessarily means that the issue has been optimally settled. While I realize that it is commercially a dead issue for now, I hope that the process of "creative destruction" capitalism is famous for may eventually yield a better result. Until then, tilting at windmills is good exercise.

Reference

This article is a re-statement and expansion of *"A Notation for Nested Arrays"* published in *"The Special Character Set, Number 3"*, the then official newsletter of the APL Special Interest Group of the Digital Equipment Computer Users Society, March 15 1982, pp. 14–16. It is unlikely to be still in print, but hopefully is not necessary for understanding this paper. The DECUS APL SIG has since merged into a Languages and Tools SIG.

Douglas R. Bohrer
Kemper Securities Inc.
77 West Wacker Drive
Chicago, Illinois 60601-1994, USA

J Locales

by Richard Oates

Explicit Definition isolates names with the Local Copula (`=.`). Locales isolate names with spelling, possibly expressed, usually implied. My trip may have been more worst case than typical. I wrote three verbs.

The name `table_bob_`, say, is `table` in locale `bob`, while `table_z_` is `table` in locale `z`. The names `table_bob_` and `table_z_` are *locatives*. One locale, the *base*, has no name. One locale, `z`, is special: its names are known in all locales unless overridden. Locales are usually populated by script. The application is scripted to the base. Secondary verbs script to named locales. If a file called `add` is:

```
table=. 1 : 0
by=. ' '&;@,.@[ ,. ]
over=. ({. ; }.)@":@,
tbl=. 1 : '[by] over x./'
 x. tbl
)
```

... then `0!:0<'\add'` scripts the adverb to the base. This scripts the adverb to locale `z`:

```
sc_z_=. 0!:0
sc_z_ <'\add'
```

I script most of my secondaries without change to locale `z`. If the first sentence of `table` is removed, and the last two as well, `by`, `over` and `tbl` become global. That is how I script my verb tracking program, and also the DOS editor I described in Vector 11.3. The former is scripted to locale `t` and the latter to locale `e`, in each case to keep names isolated by locale after Explicit (`:`) is removed.

I happened to have `cut` defined one way in the base and another way in locale `z`. Locales are populated by definition as well as script. In the next figure `5!:5@<` represents the tracking program in locale `t`. In the figure but not for real, `5!:5@<` is also defined in locale `z`. The verbs `trk` and `zing` are distinct. The first box surprised me. I expected `trk'cut'` to be `<;.1`. Then I realized locale `t` knows nothing about the base, just as the base knows nothing about `t`. After `trk_t_` is plugged into `z` it can be called `trk` but it still runs in `t`. On the other hand `zing_z_` runs in locale `z` and `zing` runs in the base.

```
cut=. <;.1
cut_z_=. <;.2
zing_z_=. trk_t_=. 5!:5@<
trk_z_=. trk_t_      NB. plug into z
d=. 1 : '(x.''cut'');(x.''cut__'');(x.''cut_z_'')'
(trk d),(zing_z_ d),:(zing d)
```

<;.2	<;.1	<;.2
<;.2	<;.1	<;.2
<;.1	<;.1	<;.2

After I saw this I added a verb to `trk` that makes a locative for any name that is not full (`cut__` from `cut`).

My DOS editor in locale `e` scripts its result to the locale determined by the extension of the file name. I substituted `locale` for `0!:0` in the definition and added two verbs. For a calendar program in locale `c`, say, the first verb makes `'sc_c_'` from `'\j\s\cal.c'`, and the second verb is almost:

```
lo=. ('locale=. '"_ ":@, ]) ; ([ ":@, '=: 0!:0'"_)
lo'sc_c_'
```

locale=. sc_c_	sc_c_=: 0!:0

For each Format (`":`) `lo` really has Do (`".`). Do is like Execute in APL. I get a nonce error if `sc_c_` is not global (`=:`).

I made no change to applications. I wrote three verbs that construct locatives for two secondaries not in `z`. All other locatives appear in a profile script. In an application session I never key a locative or see one. Names have grown up in J. Like the best people, they have lost their innocence but kept their simplicity. Locales are a major improvement.

Richard H. Oates
333 East 30 Street, Apt. 18E
New York,
NY 10016 USA

Index to Advertisers

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:

Anthony Camacho,
11 Auburn Road, Redland,
BRISTOL, BS6 6LS
Tel: 0117-973 0036
Email: acamacho@cix.compulink.co.uk

Authors wishing to use Windows Write or Word for Windows should contact Vector Production for a copy of the Vector APL TrueType font and Vector APL typebox.

Camera-ready artwork (e.g. advertisements) and diskettes of 'standard' material (e.g. sustaining members' news) should be sent to Vector Production, Brook House, Gilling East, YORK YO6 4JJ.

Tel: 01439-788385
Compuserve: 100331,644.

British APL Association: Membership Form

Membership is open to anyone interested in APL. The membership year normally runs from 1st May to 30th April, but new members may join from 1st August, November or February if preferred. The British APL Association is a special interest group of the British Computer Society, Reg. Charity No. 292,786

Name: ____________________

Address: ____________________

____

Postcode / Country: ____________________

Telephone Number: ____________________

Email Address: ____________________

Category (please tick box) to run from: 1st May ❑ August ❑ Nov ❑ Feb ❑

UK private membership	£12	❑
Overseas private membership	£14	❑
Airmail supplement (not needed for Europe)	£4	❑
UK Corporate membership	£100	❑
Corporate membership overseas	£135	❑
Sustaining membership	£430	❑
Non-voting UK member (student/OAP/unemployed only)	£6	❑

PAYMENT – in Sterling or by Visa/Mastercard/JCB only

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 Mastercard, Visa or JCB number.

I authorise you to debit my Visa/Mastercard/JCB 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)

Data Protection Act: The information supplied may be stored on computer and processed in accordance with the registration of the British Computer Society.

Signature: ____________________ Send the completed form to:

British APL Association, c/o Rowena Small, 8 Cardigan Road, LONDON E3 5HU, UK

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.

1995/96 Committee

Chairman:	Dr Alan Mayer 01792-205678x4274 a.d.mayer@swansea.ac.uk	European Business Management School, Swansea University, Singleton Park, SWANSEA SA2 8PP
Secretary:	Sylvia Camacho 0117-973 0036 100612.1057@compuserve.com	11 Auburn Road, Redland, BRISTOL, BS6 6LS
Treasurer:	Nicholas Small 0181-980 7870	8 Cardigan Road, LONDON E3 5HU
Journal Editor:	Anthony Camacho 0117-973 0036 acamacho@cix.compulink.co.uk	11 Auburn Road, Redland, BRISTOL, BS6 6LS
Activities:	Vacant Post	
Education:	Dr Ian Clark 01388-527190 100021.3073@compuserve.com	9 Hill End, Frosterley Bishop Auckland Co. Durham DL13 2SX
Technical:	Vacant Post	
Publicity:	David Eastwood 0171-922 8866 MicroAPL@microapl.demon.co.uk	MicroAPL Ltd., South Bank Technopark, 90 London Road, LONDON SE1 6LN
Recruitment:	Jon Sandles 01904-612882 100257.1756@compuserve.com	138 Burton Stone Lane, York YO3 6DF
Administration:	Rowena Small 0181-980 7870	8 Cardigan Road, LONDON E3 5HU

Journal Working Group

Editor:	Anthony Camacho	0117-973 0036
Production:	Adrian & Gill Smith	Brook House, Gilling East, YORK (01439-788385)
Advertising:	Gill Smith	Brook House, Gilling East, YORK (01439-788385)
Support Team:	Jonathan Barman (01488-648575), Duncan Pearson (01653-618900), Richard and Adam Weber (01302-539761), Sylvia Camacho, Ray Cannon (01252-874697), John Searle (0181-858 6811), David Ziemann (0181-348 4039), Jon Sandles	

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 mainframes, workstations and personal 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.

Causeway Graphical Systems Ltd
5 The Maltings, Castlegate,
MALTON, North Yorks YO17 0DP
Tel: 01653-696760
Fax: 01653-697719
Email: 100265.1564@compuserve.com

Dyadic Systems Ltd
Riverside View, Basing Road,
Old Basing, BASINGSTOKE,
Hants, RG24 0AL
Tel:01256-811125
Fax:01256-811130
Email:Sales@dyadic.com

Insight Systems ApS
Nordre Strandvej 119A
DK-3150 Hellebæk
Denmark
Tel: +45 42 10 70 22
Fax: +45 42 10 75 74
Email: insight@inet.uni-c.dk

MicroAPL Ltd
South Bank Technopark
90 London Road
LONDON SE1 6LN
Tel:0171-922 8866
Fax:0171-928 1006
Email:microapl@microapl.demon.co.uk

Dutch APL Association
Postbus 1341
3430BH Nieuwegein
Netherlands
Tel:03474-2337

Compass R&D Ltd
10 Frederick Sanger Road
Surrey Research Park
GUILDFORD, Surrey GU2 5YD
Tel:01483-302249
Fax:01483-302279

HMW Trading Systems Ltd
Hamilton House,
1 Temple Avenue,
LONDON EC4Y 0HA
Tel:0171-353 8900
Fax:0171-353 3325
Email:100020.2632@Compuserve.com

Manugistics
2115 East Jefferson St
Rockville
MARYLAND 20852 USA
Tel: +1 (301) 984-5412
Fax: +1 (301) 984-5094
Email:aplsales@manu.com (US)
Email:intl@manu.com (International)

Soliton Associates Ltd
Groot Blankenberg 53
1082 AC Amsterdam
Netherlands
Tel:+31 20 646 4475
Fax:+31 20 644 1206
Email:ljh@soliton.com