VECTOR

Meeting Reports from:

BCS®

The Journal of the British APL Association

A specialist Group of the British Computer Society

ISSN 0955-1433
www.vector.org.uk
Vol.20 No.4 April 2004

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 or an HTML document. APL code can be accepted in workspaces from I-APL, APL+Win, IBM APL2 or Dyalog APL/W, or in documents from Windows Write (use the APL2741 TrueType font, available free from the website), and MS Word (any version, ideally using the Vector template which is also available from the website).

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 2004–2005

Category	Fee	Vectors	Passes
UK Private	£20	1	1
Overseas Private	£22	1	1
(Supplement for Airmail, not needed for Europe)	£4		
UK Corporate Membership	£100	5	5
Overseas Corporate	£110	5	
Sustaining	£500	10	5
Non-voting Member (Student, OAP, unemployed)	£10	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, Vector Production, Brook House, Gilling East, YORK YO62 4JJ. Tel: 01439-788385.

Email: apl385@compuserve.com.

Contents

Guest Editorial: Is it Time for a New Name?

by Stephen Taylor

At our Annual General Meeting on 21 May we agreed to consider changing the Association's name, and to vote sometime in the coming year on a motion to do so. We did not agree on what the new name might be, but the gist of the idea is that a new name would drop 'British' and 'APL' and probably include 'Vector'.

Some members are concerned about this, quite rightly. Observers will be tempted to see this as resignation on our part to a long decline in the use of APL, and the abandonment of any hope of a recovery. They might suppose we seek a new bandwagon, or wish to disguise our old one. After two decades of commitment to APL, why change now?

Nevertheless, there are reasons to contemplate this change. Nouns depend upon the distinctions they are used to mark, much as opinions depend upon facts. Challenged once on changing his opinion, Lord Keynes famously replied, "When the facts change, I change my opinion. What do you do, sir?" Arguably, what 'APL' distinguishes has changed in the last twenty years, and is no longer quite the distinction we wish to draw.

In some ways, this is obvious. We publish in VECTOR as much about J as we do about APL, and expect to publish more and more about K. You can think of 'APL' as a kind-word, and thus J and K as kinds of APL, just as APL2 and Dyalog APL are kinds of APL. You could call on authoritative support: I have heard both Ken Iverson and Arthur Whitney refer to these languages as APLs. But you would not have the support of usage, and usage is decisive. Within the APL, J and K communities, 'APL' is used to distinguish APL2, Dyalog APL, APL2000 and their like from J and K. Theory and their authors claim they are APLs; usage has them as something else.

There used to be some debate as to whether dictionaries are descriptive or prescriptive. That debate is largely over. It is accepted-outside the Académie Française that lexicographers record usage, not dictate it. Perhaps 'APL' will one day serve as a kind-word that includes J and K. But it does not today, and a journal does better to follow usage than try to set it.

If 'APL' is no longer quite satisfactory for distinguishing the scope of our interest, it remains to see if we can find a better term. And this is the more interesting part of the whole question. What is distinctive about our languages? Our charter is to

promote their usage. What do they offer the world? The answers to these questions are not what they were two or four decades ago.

Rise and Fall

APL offered personal computing before personal computers. APL appeared as other systems were starting to share mainframe services between multiple users. It offered three distinct advantages.

- Its interpreter removed the distraction of saving, compiling, loading and running programs.
- It reacted to runtime errors by handing the user the stopped environment, with complete access to code and variables, and the ability to resume execution.
- APL's concise notation made it economical for non-programmers to write programs.

APL quickly became an underground hit among managers with access to IBM mainframes. And while APL never had a comfortable place in IBM's product range, for most of the last four decades, IBM had more employees using APL than anyone else did. Originally intended for teaching, APL spread through university programmes both as a language for programming computers, and in a variety of disciplines as a tool for modelling and exploring theories.

Two advances in personal computing, and a new movement in software development, brought this growth to a halt. The first was the appearance of VisiCalc on personal computers. With VisiCalc, Dan Bricklin presented, with Bob Frankston, a visual metaphor for the arrays he'd earlier worked on in APL. VisiCalc was a more suitable tool for the limited needs of most personal computing. Moreover, it pointed the way to the second, even more profound, advance.

The graphical user interfaces (GUIs) developed at Xerox PARC and popularised by Apple's Macintosh computers produced a dramatic shift in the economics of personal computing. A rich set of conventions, learned for the most part unconsciously by users, removed most of the cost of learning to use software. For the first time it became possible to start using new software without studying a manual or training.

The economic importance of GUIs posed a challenge to software developers. Programming GUIs is inherently difficult, blending complex conventions about the use of controls with graphic design, psychology and insight into the user's conceptualisation of the task. Formally, it requires the management of complex

states, reacting to complex and unpredictable sequences of events initiated by the user. Contemporary programming languages were inadequate for describing the solutions, and APL was no exception to this.

Xerox PARC's solution was the Smalltalk language, which describes a world of mutually opaque programmable objects communicating through messages and reacting to events. This-the 'object-oriented programming' (OOP) paradigm-is the best framework anyone has yet devised for producing GUIs.

Smalltalk had shown that the OOP paradigm could be used to address not only the essential GUIs but all computing tasks. New OOP languages such as Java and C# appeared to exploit the paradigm. Other languages adopted it, with C and BASIC becoming the C++ and Visual Basic of today.

The economic importance of GUIs has made the OOP paradigm dominant. These days computer-science graduates trained in Java and C++ can be surprised to discover that paradigms other than OOP exist.

APL is most closely associated with the Functional Programming (FP) paradigm. FP approaches programming as the application of functions to data structures. Purists eschew even naming data and have demonstrated variable-free FP solutions to complex computing tasks, much as Smalltalk programmers once demonstrated the generality of an OOP approach. However, FP has nothing to offer the GUI programmer and FP languages are uncommon in industrial programming. While the (by-now various) APLs also added object-oriented extensions for programming GUIs, APL was sidelined too.

The Software Engineering (SE) movement emerged in parallel with OOP languages. SE sought to reduce risk in software development projects by drawing on an analogy with civil engineering. It appealed to managers by offering precise information on the progress of planned work, and appealed to developers like me by proposing to impose on the chaotic development process rules as orderly as those we wrote for our machines. The most elaborate expression of SE was in the Capability Maturity Model, a fine illustration of my Latin master's description of the subjunctive-the pious wish.

Sadly, SE's analogy with civil engineering was deeply flawed, and led to the wrong conclusions. While the processes added to software development hugely improved status reporting, and generated impressive artefacts-specifications, designs, object models-they did little for productivity. Criticisms of this were met with further development processes. Now the IT divisions of many large organisations flounder in a swamp of restrictive processes. At their worst, SE

processes sacrifice productivity for a specious predictability-all that can be predicted is that the result cannot be afforded.

The heyday of Software Engineering is now perhaps passed, but in its prime, SE deprecated and expelled the fast and informal development associated with APL, or masked its productivity in expensive development procedures. When coding represents such a small fraction of the effort, why worry about programmer productivity?

Where next?

Our charter commits us to promote the use of APL. To promote its use we must be able to say what to use APL for, which is not the same as to say what it can do. In particular, we have to identify where its benefits in commercial and industrial programming outweigh the costs of including it in the tool set. If our case applies to J and K as well, then we should adopt a term - *Vector languages* - that denotes them too.

I see two applications.

Beyond Objects

In a casually incisive remark, a J tutorial observes that OOP and FP are complementary: most complex applications consist of an OOP layer on top of an FP layer. Separated from the black art of managing GUIs, APL productivity kicks butt. Recent developments such as .NET and ASP.NET have significantly lowered the cost of connecting APL processes to GUIs managed by OOP software.

Last week I was working on a web form with an ASP.NET developer when we needed to populate a drop-down list with options representing the 24 hours of the day divided into quarter hours. I had the satisfaction of writing the solution in front of him in a few minutes, testing fragments in immediate execution. (Format the hours with colons, use a catenate function table to join to the quarter hours, then ravel.) He gasped that he would have taken most of a day out to write a reusable C++ class to get the same result.

K's dictionaries and Dyalog APL's namespaces make it easy to map to and from states represented by collections of objects. OLE servers and .NET assemblies use the OOP model to let APL software plug and play with everyone else's. K's ODBC interfaces allow ASP.NET web controls to bind them.

In short, OOP tools for managing GUIs have now developed to the point where the dominance of OOP languages needs no longer be assumed. This changes the

economics of FP programming in general, and of the Vector languages in particular.

Fast and Furious

One of SE's fundamental flaws is the misidentification of programming with construction. In SE's view, smart workers break down an initial high-level design into simple coding jobs for keyboard navvies, just as architects and engineers show where to put concrete and steel. The paradigm is guaranteed to produce large teams with lots of internal communications and logistics, and inflexible projects. The software is anything but soft. In pursuit of predictability, SE cedes the real-world game of productivity from the very start.

This is not the place to expand the theme, but the Agile Development movement, of which Extreme Programming (XP) is the best-known exemplar, challenges all this. Agile Development uses small teams of smart people and fast user feedback in a work style hauntingly familiar to APL programmers. And they do it largely in C++ and Java, bless them.

In development projects where most of the effort is coding, APL productivity again becomes decisive. As team size shrinks, communication overhead drops geometrically. (Simon Garland: "A K team with two programmers is huge!") The time is ripe for combining agile methods and Vector languages.

Conclusion

APL usage has declined substantially over the last two decades. In my view, the conditions that caused this are now passing or have already passed. There are exciting new opportunities for exploiting our languages, and the implementations have never been better. The opportunities are similar for J, K and the APLs; so we will do best under a single flag. For this reason, I propose the British APL Association, a specialist group of the British Computer Society, should adopt a new name - the ***Vector Languages Group***.

A copy of this article has been posted at *www.vector.org.uk*, where you can post comments.

References

VisiCalc: http://www.bricklin.com/visicalc.htm
Functional Programming: http://homepages.inf.ed.ac.uk/wadler/guide.html
Capability Maturity Model: http://www.sei.cmu.edu/cmmi/

Quick Reference Diary

Date	Venue	Event
Nov 2004	Florida	APL2000 User's Meeting. Naples Beach Hotel, Florida. See www.apl2000.com for details
Nov 2004	Florida	Dyalog International User's Meeting See www.dyalog.com for details and online registration

Dates for Future Issues of VECTOR

	Vol.21 No.1	Vol.21 No.2	Vol.21 No.3
Copy date	In press	10th Sept	4th Dec 2004
Ad booking	In press	17th Sept	11th Dec 2004
Ad Copy	In press	24th Sept	18th Dec 2004
Distribution	August 2004	October 2004	January 2005

British APL Association News

from Anthony Camacho, Hon Sec

Minutes of the AGM of the British APL Association held at the Royal Statistical Society on 21 May 2004

The minutes of the AGM 2003 (printed in Vector) were agreed as a true record.

Adrian Smith, chairman, reported on the year. The main benefit to members is Vector; the lateness of January and April issues is regretted - Adrian ran out of steam. There is plenty of material for the April Vector and it will be going to press soon. Stephen Taylor has redesigned the web site and it now needs administrative tidying and access to the archive. Stephen asked for volunteers to help and was told Gill Smith is willing.

The meeting at the Royal Society with Arthur Whitney (combined with the twenty year celebration and the relaunch) was a success. So was the APL moot organised by Ray Cannon; so much so that Ray is thinking or running another in the autumn and some visitors from the USA are thinking about running a similar meeting on the east coast. There is also the possibility that Jim Brown may be willing to speak to us as he is visiting the UK this summer.

Lastly Adrian floated the idea of changing the name of the association to something like "The Vector Languages Group" and said this would be suggested in Vector soon; In the meanwhile *www.vectorlanguages.org* would be reserved.

The treasurer reported on the accounts and on behalf of the membership secretary. The treasurers report was accepted.

The chairman said the committee for next year would be:

Chairman	Adrian Smith
Secretary	Anthony Camacho
Treasurer	Nicholas Small
Editor	Stefano Lanzavecchia
Education	Alan Mayer
Projects	Stephen Taylor
Activities	Ray Cannon
Webmaster	Bob Hoekstra
Librarian	Morten Kromberg

Rowena Small will be membership secretary.

Summary of Annual Accounts 2003/04 (unaudited)

Summary of income and expenditure/receipts and payments:

	2003/04 (R&P) £	2003/04 (I&E) £	2002/03 (I&E) £	2001/02 (I&E) £
Income/Receipts				
Subscriptions	12460	12914	12487	9884
BCS services	0	0	0	0
Bank interest	1225	1225	1455	1703
Vector advertising (incl. VAT)	2528	1572	3432	4056
Conferences & seminars	0	0	0	0
Other	372	355	221	629
Total receipts	16585	16066	17595	16272
Expenditure/Payments				
Meetings	2451	2451	190	240
Administration	1010	860	1213	1348
BCS services	0	0	76	0
Vector production and despatch	11279	14634	16640	15352
Projects	541	541	743	2369
Conferences and seminars	0	0	0	0
Other	510	561	384	308
Total payments	15791	19047	19246	19617
Assets summary:				
Bank and other balances		36331	35537	37349
Debtors		636	1367	1121
Creditors		(8647)	(5602)	(5516)
Net assets		28321	31302	32954

Notes.

Pence figures have been omitted, so columns may not add exactly.

The value of stocks of Vector have not been assessed, nor has the value of the Association's computing hardware and software.

For 2003/04, figures are shown both as income and expenditure, i.e. revenues strictly relating to the activities of that year, and as receipts and payments, i.e. what goes in and out of our bank account. The comparative figures for earlier years relate to income and expenditure.

BAPLA membership at 30.4.04
(previous year's figures in parentheses)

	UK		FOREIGN		TOTAL	
	Number	Vectors	Number	Vectors	Number	Vectors
Sustaining	6 (6)	28 (28)	4 (2)	28 (16)	10 (8)	56 (44)
Corporate*	4 (4)	16 (16)	1 (2)	10 (12)	5 (6)	26 (28)
Corp. Ind*	9 (9)	9 (9)	15 (15)	16 (15)	24 (24)	25 (24)
Individual	106 (112)	106 (112)	209 (222)	209 (222)	315 (334)	315 (334)
Non-voting	18 (19)	18 (19)	0 (0)	0 (0)	18 (19)	18 (19)
Life	1 (1)	1 (1)	1 (1)	1 (1)	2 (2)	2 (2)
Library	1 (1)	1 (1)	6 (6)	6 (6)	7 (7)	7 (7)
Russians			11 (11)	11 (11)	11 (11)	11 (11)
APL Groups			13 (13)	39 (39)	13 (13)	39 (39)
						499 (508)

*Add the Vector numbers in these rows to get the total subscribed for by corporate members

News from Sustaining Members

MicroAPL Ltd

Following the release earlier this year of APLX Version 2.0, we now have a new-look website which includes more technical information on APLX and a new facility for buying APLX on-line for electronic download. This has proved very popular, particularly for upgrades. Because this method lowers our administrative costs, we are able to pass on savings to our customers in lower prices, in line with our goal of making APLX the most cost-effective APL system for all the major platforms.

We are also adding a new APL resource section to our website, with technical notes, utility workspaces, and links to approved APL partners. This will be rolled out progressively over the next couple of months, and will include contributions from third parties as well as from our own staff. Two items which will be of particular interest to many users are notes on how to access web pages from *APLX for Windows*, and how to interface *APLX for Linux* to the very widely-used Apache open-source web server. The second of these provides a very simple way of serving dynamic content from your APLX applications in standard web pages, without having to be concerned with the details of HTTP protocols. Apache will handle all aspects of communicating with the remote web browser, including multiple simultaneous connections, error handling, security checking, keeping a log of pages visited, etc. Your APLX application is notified via the standard CGI interface when some dynamic content (such as a price quotation) is required in the page returned to the browser.

For more details, please visit http://www.microapl.co.uk/apl

The Vector Product Guide

compiled by Gill Smith

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

We 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 Systems (Hardware & Software)
- APL and J Interpreters
- APL-based Packages
- Consultancy
- Other Products
- Overseas Associations
- Vendor Addresses
- World Wide Web and FTP Sites

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.

Your listing here is absolutely free, will be updated on request, and is also carried on the Vector web site, with a hotlink to your own site. It is the most complete and most used APL address book in the world.
Please help us keep it up to date!

All contributions and updates to the Vector Product Guide should be sent to: Gill Smith, Brook House, Gilling East, York, YO62 4JJ. Tel: 01439-788385, Email: apl385@compuserve.com

APL INTERPRETERS

COMPANY	PRODUCT	PRICES(£)	DETAILS
APL Borealis Inc.	Dyalog APL	poa	Distributor of Dyalog APL products from Dyadic
	APL2000	poa	Distributor of APL2000 products
APL Systems IDC SL	APL2000 APL interpreters and tools	poa	Distributor of APL2000 products for the UK Consulting and maintenance for APL applications
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.
Dinosoft Oy	Dyalog APL/W for Windows	poa	Finnish distributor of Dyalog APL products.
	Dyalog APL for Unix	poa	See Dyadic's listing for product details.
Dittrich & Partner	APL+Win	poa	Cognos/APL2000 Inc products
	Dyalog APL	poa	Dyadic Systems Ltd. products
	IBM APL products	poa	
Dyalog	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.
DynArray	DICE for Windows	poa	Software development kit which includes an APL interpreter as a DLL and the ability to run and link existing and new APL code to non APL code such as VB, C/C++, Java and integration with various Windows software applications and database packages such as MS Office.
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
IBM APL Products	TryAPL2	free	APL2 for educational or demonstration use. Write, fax or Email to APL Products; specify disk size desired.
	Workstation APL2 V2 Version 2	$1500	AIX, Linux, Solaris, Windows Product 5724-B74, Part Number 45P7514
	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	Cognos/APL2000 Inc	poa	Leading distributor of APL2000 products in Denmark
	Dyalog Ltd.	poa	Leading distributor of Dyalog APL products in Denmark
	IBM	poa	Leading distributor of IBM APL & GraphX products in Denmark
J Austria	J	poa	Distributor for Austria and Switzerland
	Dyalog APL	poa	Distributor
	Causeway Products	poa	Distributor
	Structural Analysis Software	poa	Complete package by IG Zenkner&Handel to perform structural analysis/engineering calculations. Also suitable for dynamic problems, e.g. earthquake simulation.
JSoftware Inc.	J on the Web online registration ...		
	J Professional (online reg.)	$895	includes manual set and one year of updates
	J Standard (online reg.)	Free	Free for download only

	Books and accessories		
	J Dictionary	$50	
	J Phrases	$50	
	J Primer	$50	
	Fractals, Visualization and J	$80	
	Concrete Math	$40	
	Exploring Math	$50	
Lescasse Consulting	APL+PC	poa	Lescasse Consulting is the exclusive APL2000 distributor in France and also distributes in Switzerland and Belgium. Call for price quotes.
	APL+Unix	poa	
	APL+DOS	poa	
	APL+Win	poa	
	Dyalog APL/W	poa	French distributor for Dyalog
MasterWork Software	Manugistics Products and ISI	poa	New Zealand distributor
MicroAPL	APLX for Windows/MacOS	499	Cross-platform APL development environment with GUI programming facilities. Interpreter modelled on APL2. Available for Windows 95/98/ME/NT/2000/XP, Mac OS 9 and Mac OS X.
	APLX Server Edition	poa	For running large multi-user APL applications on x86 Linux, RS/6000 AIX, and Windows NT/2000.
	APLX for Linux (commercial)	499	Linux desktop edition of APLX, compatible with Windows and MacOS versions, with full development environment and GUI programming facilities. Runs under RedHat, Debian, Mandrake and SuSe Linux distributions.
	APLX for Linux (personal)	Free	Full version of APLX, can be downloaded free for personal use.
Oasis	Dyalog APL	poa	Dyalog Ltd
	APL*PLUS	poa	Manugistics
	APL.68000	poa	MicroAPL Ltd
	APL2	poa	IBM
Omega	Zero	poa	A "small simple and fast" alternative to APL
Optima	Dyalog APL/W	995	Dyalog's best, leading edge APL interpreter
RE Time Tracker Oy	APL+PC (APL*PLUS/PC)	poa	Complete APL+ and Statgraphics product range and links to various 3rd party products.
	APL+DOS (APL*PLUS II)		
	APL+Win (APL*PLUS III), APL+Link		
	APL+UNIX		
	APL*PLUS Sharefile		
Soliton Associates	SHARP APL for OS/390	poa	for IBM OS/390 mainframes
	SHARP APL for UNIX	poa	for SunOS and IBM AIX
	SHARP APL for Linux	poa	for Intel Linux
Strand Software	Canada		
	All APL*PLUS products	poa	All APL*PLUS products including upgrades and educational.
	Dyalog and JSoftware products	poa	
	USA		
	Dyalog and JSoftware products	poa	

APL PACKAGES

COMPANY	PRODUCT	PRICES(£)	DETAILS
ADAPTA Software	MPS	poa	Master Production Scheduling
	FBS	poa	Forecasting and Budgeting System
	DRP	poa	Distribution Requirements Planning
Adaptable Systems	FLAIR	poa	Finite loader and interactive rescheduler. Customisable full-function scheduling system. (Available outside Australia by special arrangement only.)
Adaytum	Adaytum e.Planning	poa	Adaytum e.Planning offers a Web-based solution that combines planning, forecasting, budgeting, modelling and reporting in a single, integrated application.
APL Software\Services	APL Utilities	poa	Software: mostly .AWS for DOS, utilities for most APL interpreters. Public domain APL*Plus v10 with on-screen documentation and interactive tutorials. APL Conference Software. Books: APL user manuals for STSC, IBM, and Sharp. Request email catalog from dick_holt@email.com.
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).
	SF_READ	poa	Dyalog APL/W functions to read APL*PLUS data objects of any type or structure from *.SF style component files created by APL*PLUS II or III.
Causeway	*CausewayPro* for Dyalog/W	400	Causeway application development platform for Dyalog APL/W.
	RainPro Business Graphics	250	The ultimate graphics toolkit for the APL developer. Adds 3D charting capability, Web publishing and clipboard support to the shareware product. Charts can be included in *NewLeaf* reports. Functionally compatible across Dyalog/W and APL+Win.
	NewLeaf for Dyalog and +Win	400	Frame-based reporting tool with comprehensive table-generation and text-flow support. Offers multiple master-page capability, bitmap wrap-around and on-screen preview with pan and zoom. Fully supported on Dyalog/W and APL+Win
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	3-way TANGRAM	poa	3-way TANGRAM is a DSS-OLAP product, basically a powerful and versatile handler of multi-way tables (also known as hypercubes). It entails 46 analysis modules, including computations, cube merging, sensitivity analysis, time intelligence, free format queries, HTML and LaTeX outputs. Current version is 7.0 At the time of writing, the product is available on Dyalog APL 8.3.1 for Windows 95/98/NT. Future plans include an APL+WIN version and later a LINUX version.

CoSy	K.CoSy	$30 % yr sub	K.CoSy is a general purpose computing and programming environment constructed, all in open code, in Arthur Whitney's very high level, yet structurally transparent Array Programming Language, K , and its tightly coupled User Interface. K.CoSy is an extremely productive environment in one of the most powerful and fastest of APL's progeny, and therefore, likewise, of all languages. K.CoSy provides a workspace-like interactive development environment previously impossible in K. Because of its unique open construction within the language itself, this environment is clearly competitive in a large domain with the APLs from the other vendors. K.CoSy notepad nature, interactivity, and open K code vocabulary make learning Arthur Whitney's K far less daunting and far more productive than its raw console, or any external scripting method. If you are a client of Kx Systems , or are investigating the possibilities, Contact us. See CoSy/K/CoSy for more information.
DynArray	DynaWeb Server	poa	A web server providing web based access to applications running on the DICE interpreter from DynArray, or on an IBM mainframe running APL2.
	DynaHarry	poa	A DSS system which offers the next generation capabilities for current APLDI, IC/E and IC/1 users. It comes with ROLAP capabilities, multisystem access to a wide variety of databases and data warehouses.
	DynaLink	poa	An ODBC client interface for DICE and IBM APL2 programs.
Entropy Software	Personal Information Management	poa	Entropy specialises in Personal Information Management Systems (PIMS) which run on a PC. Most systems are inadequte in some way. This is an attempt to fulfil all the needs of George MacLeod who has been an unhappy user of a variety of systems for many years. The product is being developed in modules (Calendar, Address List, To Do List, etc) which will be available at no charge. When all the modules are complete they will be integrated together. There will be a charge for the integrated product.
HMW	4XTRA	poa	Networked, Windows/Unix based Front End and Middle Office Foreign Exchange and Money Market Dealing System. Scalable from 1 user to 120+.
	Inca	poa	Software Change Management System. Enables the user to co-ordinate development work from several sources, resolve clashes, promote work items for testing and configure releases to a live environment.
	Maya	poa	APL code file manager. A comprehensive suite of tools giving a multi-window IDE style interface to file based APL code. Offers features such as copying from file to file, object comparison, string search, style formatting, hot-spot editor for filed objects (including variables), etc.
	Aztec	poa	System shell for APL development. Manages real-time and batch applications across multiple platforms. Offers standardised error trapping, job scheduling, task communication and recovery/restart features.
	Olmec	poa	APL GUI environment, providing menu bar, tool bar, status area, navigation sidebar (with treeviews & listviews)and client area. All are configured by simple text files and require no programming. Client area has a "tab wizard" option to provide ordered transaction processing
	Nazca	poa	fast, flexible and reliable static database and editor.
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!
Insight Systems	Causeway	poa	Leading distributor of Causeway products in Denmark

All our old products are now either OEM'd, in the public domain, out of date, or all of the above. We'll be back!

JAD Software	JAD SMS	poa	JAD SMS is a multi-user software management system for Dyalog APL™ based on shared, hierarchical databases. JAD SMS databases let you keep historical versions of apl items as well as attributes such as timestamp, user name and documentation. The software includes a graphical user interface as well as specialized functions for inclusion in applications. No charge for single-user version; $100/user for multiple users
Lescasse Consulting	APL+Win Monthly Training	$600	Download 50+ page document about APL+ programming each month. You also get one or more workspaces full of re-usable APL code and sometimes additional files or products.
	Advanced Windows Programming	$95	200-page book plus companion disk on interfacing APL and Delphi. Contains full coverage of Delphi-2, +Win and Dyalog.
	DLL parser for APL	$250	Parse any Visual Basic DLL declaration file into a set of quadNA definitions. Turn constants and structures into APL variables. Available for APL+Win and Dyalog/W.
	Delphi Forms Translator	$195	Design forms with Delphi and turn them automatically into APL programs which recreate the same form (+Win and Dyalog/W).
	APL+Link Pro	poa	ODBC interface for APL+Win
	SQAPL Pro	poa	ODBC interface for Dyalog APL/W
	RainPro	poa	Highly customisable 2D and 3D publication graphics for APL+Win and Dyalog APL/W
	NewLeaf	poa	Page layout and printing tools for APL+Win and Dyalog
	GraphX and ChartFX	poa	High-quality business graphics for APL+Win
	Formula One and Dyalog APL	$95	100-page book + companion disk on how to use the Formula One VBX with Dyalog APL/W
Lingo Allegro	FACS	poa	EMMA-like interface to DB2 or ODBC databases
	QWIN	poa	Legacy DOS Windowing support for APL+Win
	ODBC/127	poa	IBM AP127-like ODBC Interface for APL+Win and Dyalog APL/W
Optima	ServiceLine	poa	A property management system which keeps a record of all outstanding tasks, produces an up to date list of work to-do and Scheduled reminders plus automated standard letters and basic financial control.
	TravelLine	poa	A system designed to control the workload allocation for a fleet of chauffeur vehicles plus a reminder system for fleet management and Local Authority requirements.
	BPA	poa	"Brand Performance Analysis" allows for the modelling of product/brand performance over time and comparison with competitor products.
	DBI	poa	"Database Interrogator" allows for the non technical user to ask sensible questions of a large database such as a questionaire and obtain results tables and graphs quickly, easily and accurately.
Qualedi	Qualedi	$850-$5,500	Electronic Data Interchange (EDI) translation software for the PC, with strict compliance checking.
RE Time Tracker Oy	UIT/W	poa	Comprehensive high-level Windows User Interface library for APL+Win and +II v 5.1. Comprehensive spreadsheets, replicated fields, special field types, etc. 16 and 32 bit versions available.
	AJGRAPH	poa	Graphpak-compatible 2D graphics package for +Win and +DOS. Includes multi-window support, print and metafile support. No DLLs required.
	ECCO PRO with APL	poa	Leading group and personal information management system with comprehensive customising. Supplied with sample +Win workspace to interface to ECCO databases via DDE.
	NEWT TCP/IP SDK with APL	poa	Lead TCP/IP SDK with interfaces to all protocols. Supplied on 3 CD ROMS together with a sample +Win workspace.

	DB+	poa	Database interface for APL+DOS under Windows. Allows combining character-based APL applications with ODBC-compliant databases such as Oracle and SQL-server.
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.
Weighahead Systems	Weighahead Windows Weighing System (3WS)	poa	Recipe Weighing System for Manufacturing Industries. Pharmaceutical, Cosmetics, Foods etc. Works without keyboard or mouse. Uses Electronic Balances, Laser scanners, bar codes and label printers.
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 AND DEVELOPMENT

COMPANY	PRODUCT	PRICES(£)	DETAILS
Adfee	Consultancy	poa	Development, maintenance, conversion, migration, documentation, of APL products in all APL environments
Andrews	Consultancy	poa	APL programming and analysis, algorithms, tree processing and design programs for craft work.
APL Borealis Inc.	Support and Development	poa	APL Software Support and Development. Specialists since 1979 in Sharp APL, APL*Plus, APL+Win, Dyalog APL
APL Solutions Inc	Consultancy	poa	APL systems design, development, maintenance, documentation, testing and training. Providing APL solutions since 1969.
APL Systems IDC SL	Consultancy	poa	Consulting and maintenance for APL applications.
AUSCAN Software	Consultancy and Training	poa	APL software development, training
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
Ian Clark	Consultancy	poa	Interfacing APL, VB, C/C++, ActiveX/COM. Screen design and documentation. National language porting.
Ray Cannon	Consultancy	poa	APL, C, Assembler, Windows, Graphics: PC and mainframe
Causeway	Consultancy and Training	poa	On-site training for Causeway, RainPro and NewLeaf. Customisation and enhancement to meet local needs. Code review and pre-implementation check of Causeway applications.
Paul Chapman	Consultancy	250-500	24-hour programmer: APL, Smalltalk, C; Windows front end design a speciality.
CODEWORK	Consultancy	poa	Development, maintenance, migration, documentation of APL applications. Speciality: info systems for top executives, internet applications.
CoSy	Consultancy	poa	CoSy.com, Coherent Systems, provides rapid development in the K language and associated data base products, with a particular interest in quantitative (financial) modelling.
David Crossley	Consultancy	poa	Experienced in large APL system developments since 1969 for PC or mainframe.
Dinosoft Oy	Consultancy	poa	Specialised in very large databases.

Dittrich & Partner	Consultancy	poa	APL programming and analysis; APL workshops and training on the job
Dyalog	Consultancy	poa	APL and Unix system design, consultancy, programming and training.
DynArray	Consultancy	poa	DynArray offers consulting in the areas of DSS, Y2K and APL programs upgrade/conversion to modern Web enabled platforms.
Evestic AB	Consultancy	poa	Excellent track record from 15+ years of APL applications in banking, insurance, and education services. All dialects, platforms and project phases. SQL expertise.
First Derivative Analytics Ltd.	Consultancy	poa	Analysis, design, prototyping, development & testing of APL (especially financial) applications: Sharp, Dyalog APL/W.
General Software	Consultancy	from 200	Over 20 years experience with every version of APL, large mainframe systems and small PC based programmes.
Godin London Inc	Software Development	poa	We have applications in the food manufacturing field, travel agency and airline bookings field and in product lease management.
HMW	Consultancy	poa	System design consultancy, programming. HMW specialize in banking and prototyping work.
Hoekstra Systems	Consultancy	poa	APL consultancy, programming, etc. Also UNIX system administration
Michael Hughes	Consultancy	poa	APL consultant with 20 years experience with all versions of APL. I can create your dynamic Web sites using the full power of APL working with Microsoft IIS (Internet Information Service) on Windows NT or 2000. I also undertake System design, Programming and Maintenance on all platforms, particularly MS Windows.
INFOSTROY	Consultancy	poa, competitive	Broad experience in various APL platforms. Special skills and knowledge in developing complex applications for investment, financial and construction markets. Implementation of hybrid solutions based on APL, Delphi, C++, VBA, SQL servers.
Insight Systems	Consultancy	poa	We have experience with just about every APL system and platform in common use during the last 20 years, from SHARP APL under MVS or Linux to APL+Win and in particular Dyalog APL under Windows 9x, NT or 2000. If you have decisions to take about adapting your APL application to take advantage of emerging technologies, or would like your strategy reviewed, give us a call. We have extensive experience in all areas of APL development, from legacy systems, up, down and sideways migrations, to the development and support of shrink wrapped solutions based on APL. Even if we don't have time to do the work ourselves, we will know where to find someone who is an expert in your version of APL and your application area, on your continent.
JAD Software	Consultancy	poa	Systems design and development, project management, technical manuals, financial and actuarial expertise in APL.
KJK	Consultancy and software development	poa	APL-based data management: conversions, ad hoc-analyzing tools, well-interfaced methods for defining, processing and browsing of multi-dimentional reports. Rapid custom software development based on proven modular toolset approach.
Lambent Technolgy	Consultancy	poa	APL programming, consulting & training; web design and construction.
Phil Last	Consultancy	poa	APL consultancy, modelling and programming.
Lescasse Consulting	Consultancy	poa	A range of consultants, experts in Windows programming, with APL+Win and Dyalog APL/W. More than 100 major APL applications already developed. We all have additional expertise in Formula One and Delphi.
Lingo Allegro	Consultancy	poa	General APL consulting, internet website development, migration and downsizing, performance tuning, education and training.
Lucas Solutions	Consultancy	poa	Rates depend on task and location.

Mackay Kinloch Ltd	Consultancy	poa	Design, analysis and programming for banking, insurance and pensions, financial planning and modelling, corporate performance and legal reporting
MasterWork Software	Consultancy	poa	Consulting and J programming for econometrics and statistics in public policy, health and food industries.
Milinta Inc	Consultancy	poa	Design, development, maintenance, conversion, documentation in all APLs, most APs and some specific Sharp products (LOGOS, ViewPoint, Retrieve). Experience in multi-user, multi-task systems, databases, Windows programming.
Ellis Morgan	Consultancy	poa	Business Forecasting & APL Systems.
Oasis	Consultancy	poa	Expertise in APL system design, Project management, conversion, migration, tuning; for all APL versions (10+ years experience)
Omega Computing	Consultancy	poa	APL consultancy, programming, etc.
Optima	Consultancy, Training & Development	poa	We have been in business since 1990 and have a team of APL professionals with many years experience in pharmaceutical, industrial and financial systems on both PC and Mainframe platforms. In addition to pure APL we can also offer assistance with network integration, ACT! customisation and issues concerning the internet and/or web design.
RadSys Technologies	Consultancy	poa	Areas of expertise: financial systems, risk analysis systems, healthcare systems.
Resources & Results	Consultancy	poa	Knowledge management company builds decision support, data warehouse, data mining and strategic planning systems for CFO's, CEO's, and senior management, using our Rapid Application Development (RAD) methods and tools. Extensive experience in large-scale system development and ad hoc executive support for Fortune 500 clients.
RE Time Tracker Oy	Consultancy	poa	APL application conversions, APL Windows interfaces, APL to API-level interfacing to any system under Windows, TCP/IP network and database connectivity.
Graeme Robertson	Consultancy & Training	250-500	Design and write custom software. Maintain and upgrade APL systems. Deliver customized APL training courses
Rochester Group	Consultancy	poa	Specialise in MIS using Sharp APL
Shepp & Associates	Consultancy	poa	APL applications development and consulting, especially in the travel industry, especially on small computers. 25 years experience in APL programming.
Snake Island Research Inc	Consultancy	poa	APL interpreter and compiler enhancements, intrinsic functions, performance consulting. APL parallel compiler APEX is giving very good initial performance tests with convolution somewhat faster than FORTRAN.
SovAPL	Consultancy	poa	Offshore APL development service.
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.
Rex Swain	Consultancy	poa	Independent consultant, 25 years experience. Custom software development, PC and/or mainframe.
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.
		MACfns	MACfns has been released. A suite of over 100 high-accuracy, high-speed assembler code utility functions for APL+Win and APL+DOS, MACfns has been thoughtfully designed and implemented for APL programmers. It includes functions for segmented string manipulation, data restructuring, translation and datatype adjustment, and many numerical computations.Typical speedups range from 4-10 times, with reduced workspace usage, datatype conservation, identity detection, and greater numerical

			precision amongst the other benefits. Documentation of MACfns is extremely detailed and thorough.
Weighahead Systems	Consultancy	poa	Specialising in industrial systems. Links to PLCs, laser scanners, bar codes, weigh scales, label printes etc. Also programmable hand held scanners.
Stephen Wynn	Consultancy	poa	Most experience of financial planning, and mathematical areas: operational research, quality control, experimental design.

OTHER PRODUCTS

COMPANY	PRODUCT	PRICES(£)	DETAILS
Adfee	Employment	poa	Contractors and permanent employees
APL-385	Typefaces	poa	Variants of the APL2741 typeface available to specification.
APL Borealis Inc.	APL Training	poa	Hands-on courses in Introductory, Intermediate, Advanced and Windows APL. Courses are customized and flexible, and may be delivered on-site, with strong emphasis on methods for efficient and maintainable APL systems development.
Atomic Vector	RELCELL	poa	Details: Report Writer, Data Mining, Database Visualization, and Export to Excel from relational databases. Offered in English, Spanish, Italian, Portugese, French and German. With this product, you don't have to know English in order to use SQL. Point and click with easy to use Wizards.
	APL Games Volume 1	$15	Details: Volume 1 - A collection of four APL games on cd, saved from past yearly issues of the APL Planet. Great way to demonstrate the capabilities of APL, and to entice the younger generations into APL. Written entirely in Dyalog APL.
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.
I-APL Ltd	Books	poa	I-APL stocks books written to go with the I-APL Interpreter and some APL Press books. For a list write to 11 Auburn Road, Bristol BS6 6LS, ring 0117 973 0036 or email acam@blueyonder.co.uk.
Oasis	Training	poa	Introductory courses in APL Advanced courses for different APL versions
Optima	IBM Compatible	poa	Complete business solutions including hardware, software and support plus additional network installation and maintenance if required.
Renaissance Data Systems	Booksellers		The widest range of APL books available anywhere. See Vector advertisements.
Right Seat Software	Vox Proxy	$199 (comm) $69.95 (edu)	Vox Proxy is authorware for PowerPoint(r) 2000 or 2002 which allows the use of Microsoft Agent Technology (3D talking animated characters) within slide shows. VP appears on PowerPoint's main menu and provides editing side-by-side with slides. Automated script-writing provides control of PowerPoint, allowing the use of characters for live presentations or fully-automated tutorials, demos, or training programs. Optional CD Prep program allows the user to create auto-starting CD's that will play on any version of PowerPoint or without PowerPoint.

OVERSEAS ASSOCIATIONS

GROUP	LOCATION	JOURNAL	OTHER SERVICES	Ann.Sub.
ACM SigAPL	International	APL QuoteQuad	Conferences; APL white pages; web site	$30
APL Bay Area	USA N. California	APLBUG	Monthly Meetings (2nd Monday)	$20
APL Club Austria	Austria	-	Quarterly Meetings	200AS(Indiv), 1000AS(corp)
APL Germany e.V.	Germany	APL Journal	Semi-annual meetings	DM60

Ass. Francophone pour la promotion d'APL	France	Les Nouvelles d'APL		FF350 (private) FF2800 (Company)
BACUS	Belgium	APL-CAM	Conferences & Seminars	£18 ($30)
Danish SIG	Denmark			
Dutch APL Assoc.	Holland	Vector provided	Mini-congress, APL ShareWare Initiative	
FinnAPL	Helsinki, Finland	FinnAPL Newsletter	Seminars on APL	100FIM(private), 30(student), 1000 (Co)
Japan APL Assoc	Tokyo	APL Journal	Monthly meetings (4th Sat)	10,000yen to join
NY SigAPL	New York, USA	Big Apple APL	Monthly meetings	$35/$25(ACM)
Rome/Italy SIG	Roma, Italy			
SE APL Users Grp	Atlanta, Georgia	The APL Planet	Semi Annual meetings	$13 (US), $25 (non-US)
SovAPL	Moscow, Russia	-	Seminars and Annual Meeting	
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)
Toronto SIG	Toronto, Canada		Occasional meetings, APL Skills Database, Toronto Toolkit	

ADDRESSES

ORGANISATION	CONTACT	ADDRESS, TELEPHONE, FAX, EMAIL etc.
ACM SigAPL	David Siegel	ACM, 1515 Broadway, 17th Floor, New York, NY 10036, USA (Subs only)
ADAPTA Software GmbH	Michael Baas	Stellinger Weg 19, 20255 Hamburg, Germany. Tel: +49 40 40170951 Fax: +49 40 40170952. Email: info@adapta.de
Adaptable Systems	Lois & Richard Hill	49 First Street, Black Rock 3193, Australia. Tel: +61 3 9589 5578 Fax: +61 3 9589 3220 Email: hillrj@melbpc.org.au
Adfee	Bernard Smoor	Dorpsstraat 50, 4128 BZ Lexmond, Netherlands. Tel +31 347 342 337 Fax: +31 347 342 342 Email: adfee@concepts.nl
Andrews	Dr Anne D Wilson	12 Thorny Hills, Kendal, Cumbria LA9 7AL, UK. Tel: 01539-731205 Email: ADWilson@kencomp.net
APL-385	Adrian Smith	Brook House, Gilling East, York YO62 4JJ, UK. Tel: 01439-788385 Email: apl385@compuserve.com
APL2000 (Europe)	Fred Honea	see APL Systems IDC SL.
APL Bay Area APLBUG	Curtis Jones (Sec)	228 South 15th Street, San Jose, CA 95112-2150, USA Tel: +1 (408) 292-4060 Email: jonesca@vnet.ibm.com
APL Borealis Inc.	Richard Procter	381 Manor Road East, Toronto, Ontario M4S 1S7, Canada. Tel: (416) 457-7828. Fax: (416) 482-6582 Email: info@aplborealis.com
APL Club Austria	Harald F. Nelson	c/o N-TECH, Siebenbrunnenfeldg. 4-6, A-1050 Wien, Austria. Tel: +43 1 5458063 Fax: +43 1 5458063-17
APL Germany e.V.	Dieter Lattermann	Rheinstraße 23, D-69190 Walldorf, Germany. Tel: +49 6227-63469. Email: Dieter_Lattermann@t-online.de
APL Software\Services	Dick Holt	31570 Heather Field Lane, Earlysville VA 22936 USA Tel: +1 (434) 973-4255; Fax: +1 (703) 528-7617; Email: dick_holt@email.com
APL Solutions Inc	Eric Landau	1107 Dale Drive, Silver Spring, MD 20910-1607 USA Tel: +1 (301) 589-4621 Fax: +1 (301) 589-4618 Email: aplsi@starpower.net
APL Systems IDC SL	Fred Honea	Alfredo Marquerie, 12 - 2 F, 28034 Madrid, Spain. Tel: +34 91 730 7008 (Office) +34 60 680 5949 (Mobile). Fax: +1 775 743 6131. Email: uksales@apl2000.net
Association Francophone pour la promotion d'APL	Ludmila Lemagnen	174 Boulevard de Charonne, F-75020 Paris, FRANCE Email: lemagnen@aol.com
AtomicVector Productions Inc.	John Manges	413 Comanche Trail, Lawrenceville, GA 30044, USA. Tel: +1 (770) 972-3755 Email: E-mail: mailto:seapldoc@aol.com
AUSCAN Software Ltd	Richard Procter	PO Box 39, Mansfield, Ontario L0N 1M0 Canada Tel: +1-705-434-1239 Email: rjp@ca.inter.net

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 Email: BeautifulSystems@goffs.net
Camacho	Anthony Camacho	11 Auburn Road, Redland, Bristol BS6 6LS, UK. Tel: 0117-973 0036. email: acam@blueyonder.co.uk
Ray Cannon		21 Woodbridge Rd, Blackwater, Camberley, Surrey GU17 0BS, UK. Tel: 01252-874697 Email: ray_cannon@compuserve.com
Causeway Graphical Systems Ltd	Adrian Smith	The Maltings, Castlegate, MALTON, North Yorks YO17 7DP, UK. Tel: 01653-696760 Fax: 01653-697719 Email: adrian@causeway.co.uk
Paul Chapman		51B Lambs Conduit Street, London WC1N 3NB, UK. Tel: 020 7404 5401. Compuserve: 100343,3210
Cinerea AB	Rolf Kornemark	Box 61, S-193 00 Sigtuna, Sweden. Tel/Fax: +46 859 255 421 Email rolf@cinerea.se
Ian Clark	Ian Clark	1205 Deer Creek Drive, Plainsboro, NJ 08536, USA. Tel: +1 609 716 8832 Email: earthspot2000@hotmail.com
CODEWORK	Mauro Guazzo	Corso Lanza 105, 10133 Torino, Italy. Tel: +39 11 6601560 Fax: +39 11 6601560 Email: codework@codework-it.com
ComLog Software	Jeff Pedneau	18728 Bloomfield Road, Olney, MD 20832 USA Tel: +1 (301) 260-1435 Email: jeff@softmed.com
CoSy.com	Bob Armstrong	42 Peck Slip #4B, New York, NY 10038-1725, USA. Tel: +1 212-285-1864 Fax: +1 212-285-1864. E-mail: bob@CoSy.com.
David Crossley	David Crossley	187 Le Tour du Pont, 84210 ST DIDIER, France. Tel: +33.4.90.66.08.87 Email: crossley@au-village.com
Danish User Group	Helene Boesen	c/o Insight Systems ApS, Nordre Strandvej 119G, Hellebæk, Denmark
Dinosoft Oy	Pertti Kalliojärvi	Lönnrotinkatu 21C, 00120 Helsinki, FINLAND. Tel: +358 9 70028820 Fax: +358 9 70028824 Email: dinosoft@dinosoft.fi
Dittrich & Partner Consulting GmbH	Axel Holzmüller	Kieler Strasse 17, D-42697 Solingen, Germany. Tel: +49 212-260 660 Fax: +49 212-260 6666; Email: info@dpc.de
Dutch APL Association	Bernard Smoor (Sec)	Postbus 1341, 3430BH Nieuwegein, Netherlands. Tel: +31 347 342 337 Fax: +31 347 342 342
Dyalog Ltd.	Peter Donnelly	Grove House, Lutyens Close, Chineham Court, Basingstoke, Hampshire RG24 8AG, UK. Tel: 01256-338461 Fax: 01256-316559 Email: sales@dyalog.com; support@dyalog.com
DynArray Corporation	Dr James Brown	16360 Monterey Rd. Suite 260, Morgan Hill, CA 95037, USA Tel: +1 (408)-782-6648 Fax: +1 (408)-782-6627 Email:info@DynArray.com
Entropy Software Limited	George MacLeod	Calle Viena 13, Residencia Larrosa, Guardamar del Segura, 03140 Alicante, Spain. Tel: +34 61 00 41 940 Email: george.macleod2003@yahoo.co.uk
Evestic AB	Olle Evero	Bertelsusvagen 12A, S-146 38 Tullinge, Sweden Tel & Fax: +46 778 4410 Email: olle.evero@mail.com
FinnAPL	Olli Paavola	Suomen APL-Yhdistys RY, FinnAPL RF, PL 1005, 00101 Helsinki 10, Finland Email: olli.paavola@pyr.fi
First Derivative Analytics Ltd.	Ken Chakahwata	114 Lemsford Lane, Welwyn Garden City, Herts AL8 6YP, UK Tel/Fax: 01707-339620. Email: KenChakahwata@compuserve.com
General Software Ltd	M.E. Martin	Little Wester House, Westerhill Road, LINTON, Kent ME17 4BS Tel: 01622 832463 E-mail: martin@gsoft.plus.com
Godin London Incorporated	Gaëtan Godin	12 Gerrard St., London, Ontario, Canada N6C 4C5 Tel: +1 (519) 679-8290 Fax: +1 (519) 438-6381 Email: info@godin.on.ca
HMW Computing	Chris Hogan	Hamilton House, 1 Temple Avenue, London EC4Y 0HA, UK. Tel: 0870-1010-469; Email:HMW@4xtra.com
Hoekstra Systems Ltd	Bob Hoekstra	Dominique, Salisbury Road, Woking, Surrey, GU22 7UR, UK. Tel: 01483-771028. Fax: 01483-837324 Email: Bob.Hoekstra@HoekstraSystems.ltd.uk
Michael Hughes		28 Rushton Road, Wilbarston, Market Harborough, Leics. LE16 8QL, UK. Tel: 01536-770998 Email: Michael@Hughes.uk.com
I-APL Ltd	Anthony Camacho	11 Auburn Road, Redland, Bristol BS6 6LS, UK. Tel: 0117-973 0036. Email: acam@blueyonder.co.uk

IBM APL Products	Nancy Wheeler	APL Products, IBM Silicon Valley Lab, Dept H36/F40, 555 Bailey Avenue, San Jose CA 95141, USA. Tel: +1 (408) 463-APL2 [+1 (408) 463-2752] Fax: +1 (408) 463-4488 Email: APL2@vnet.ibm.com
INFOSTROY	Alexey Miroshnikov	3 S. Tulenin Lane, St. Petersburg 191186 Russia. Tel:+7 812 325-9797 Fax:+7 812 311-2184 Email:aim@infostroy.ru
Insight Systems ApS	Helene Boesen	Nordre Strandvej 119G, DK-3150 Hellebæk, Denmark Tel:+45 70 26 13 26 Fax: +45 70 26 13 25 Email: info@insight.dk
JAD Software	David Crossley	175 East 96th St., Apt. 17G, New York, NY 10128 Country: USA Tel: +1 (212) 369-6713 Fax: +1 (212) 761-0124 Email: jadsms@usa.net
Japan APL Assoc	Toshio Nishikawa	1-8-13 Masujima Buld.6F Higashi Gotanda Shinagawa-ku, Tokyo Japan 141-0022. Tel: +81 (03) 3280-0411 Fax: +81 (03) 3280-0418 Email: KYY00361@niftyserve.or.jp
J Austria	Joachim Hoffmann	Hohenllz Nord 104, 8182 Puch bei Weiz, Austria. Tel: +43 (0)699 191 42 512 Mobile: 0043 (0)699 1 91 42 51 2. Email: joachim.hoffmann@gmx.at
JSoftware Inc.	Eric Iverson	33 Major Street, Toronto, Ontario, Canada M5S 2K9. Tel: +1 (952) 470-7345 Fax: +1 (952) 470-9202 Email: info@jsoftware.com
KJK-tieto Oy	Kimmo Kekäläinen	Merikasarminkatu 10 B 56,00160 Helsinki, Finland. Tel: +358 50 55 27 207; Email: Kimmo.Kekalainen@pp.htv.fi
Kx Systems	Simon Garland [Europe]	555 Bryant St., No. 375. Palo Alto CA 94301 USA. Tel: +1 866 kdb fast email: info@kx.com (in Europe simon@kx.com)
Lambent Technology Ltd	Stephen Taylor	81 South Hill Park, London NW3 2SS, UK. Tel: +44(0)20 7813 3786. Email: sjt@lambenttechnology.com
Phil Last Ltd	Phil Last	146 Crossbrook Street, Cheshunt, Herts, EN8 8JY, UK. Tel: 01992-633807 Fax: 0121-359 0375 Email: phil.last@net.ntl.com
Lescasse Consulting	Eric Lescasse	18 rue de la Belle Feuille, 92100 Boulogne, France. Tel: +33.1.46.05.10.76 Fax: +33.1.46.04.60.23 Email: eric@lescasse.com
Lingo Allegro USA, Inc	Walter G. Fil	203 N. LaSalle Street, Suite 2100, Chicago, Illinois 60601, USA. Tel:+1 (800) 546 4621 E-mail: lingo-allegro@visto.com
Lucas Solutions	Jim Lucas	Stubbedamsvej 9C, 3.tv., 3000 Helsingør, Denmark Tel: +45 49 26 52 42. Email: jel@danbbs.dk
Mackay Kinloch Ltd	Alastair Kinloch	519 Webster's Land, Edinburgh EH1 2RX, Scotland, UK. Tel: +44 (0)7802 430 202 Email: alastair.kinloch@btinternet.com
MasterWork Software Ltd	Fraser Jackson	PO Box 56-036, Tawa, Wellington, New Zealand. Tel: +64 (4) 232-4440 Fax: +64 (4) 232-4452 Email: fraser.jackson@xtra.co.nz
Mercia Software Ltd.	Gareth Brentnall	Mercia House, Ashted Lock, Aston Science Park, Birmingham, B7 4AZ, UK. Tel: 0121-359 5096. Fax: 0121-359 0375
MicroAPL Ltd.	Richard Nabavi	The Roller Mill, Mill Lane, Uckfield, E.Sussex TN22 5AA Tel: 01825 768050. Fax: 01825 749472 Email: MicroAPL@microapl.demon.co.uk
Milinta Inc.	Dan Baronet	Contact Dan Baronet at dbaronet@milinta.com
Ellis Morgan	Ellis Morgan	Myrtle Farm, Winchester Road, Stroud, Petersfield, Hants GU32 3PE, UK. Tel: 01730-263843 Email: apl@ellismorgan.co.uk
NY Sig	David Siegel	PO Box 2697; New York, NY10163-2697, USA. Email: NYSIGAPL@ACM.ORG
Oasis b.v.	Louis Rijkse	Lekstraat 4, 3433 ZB Nieuwegein, Holland. Tel: +31 30 60 66 336 Fax: +31 30 60 65 844 Email: rijkse@oasis.nl
Omega Computing Inc	Alan Graham, Andrew Chou	3 Columbus Avenue, Edison, NJ 08817, USA. Tel: +1 (732) 985 9519 Email: alangraham@mindspring.com
Optima Systems Ltd	Paul Grosvenor	Optima House, Mill Court, Spindle Way, Crawley, West Sussex, RH10 1TT, UK. Tel: 01293 562700 Fax: 01293 562699 Email:paul@optima-systems.co.uk
Qualedi Inc.	Nicole Schless Georges Brigham	121 West Main Street, Milford, CT 06460, USA. Tel: +1 (203) 874-4334 Fax: +1 (203) 876-9083. Email: sales@qualedi.com; info@qualedi.com
RadSys Technologies AB	Randolph Schrab	Lovsangarv. 18, S-756 52 Uppsala, Sweden. Tel: +46 18 32 41 53 Fax: +46 708 1996 11 Email: randolph.schrab@radsys.se
Renaissance Data Systems	Ed Shaw	P.O. Box 511, Botsford, CT 06404, USA. Tel: +1 (203) 270-9729 Email: aplbooks@earthlink.com
Resources & Results	Frank Rhodes	342 Glen Oaks Blvd. Dallas, TX 75232, USA. Tel: +1 (817) 501-9825 Email: frank21@decision-support.net
RE Time Tracker Oy	Richard Eller	Mikonkatu 8 A, 2.krs, PL 363, 00101 Helsinki, Finland. Tel: +358 9-621 3300 Fax: +358 9-621 3378 Email: re@rett.fi

Right Seat Software, Inc.	Tom Atkins	1110 12th Street, Golden, CO 80401, USA. Tel:+1 303 278 2244. Fax: +1 303 278 6967. Email: info@voxproxy.com
Graeme Robertson	Dr. Graeme D. Robertson	15 Little Basing, Basingstoke, Hants RG24 8AX, UK. Tel: +44 (0) 1256-364071 Email: GraemeDR@nildram.co.uk
The Rochester Group Inc.	Robert Miller	600 Park Avenue, Rochester, NY 14607-2926, USA. Tel: +1 (716) 271-1110. Fax: +1 (716) 271-1230
Rome/Italy SIG	Mario Sacco	Casella Postale 14343, 00149-Roma Trullo, Italy Email: mario.sacco@tin.it
SE APL Users Group	John Manges	413 Comanche Trail, Lawrenceville, GA 30044, USA Tel: +1 (770) 972-3755 Email: seapldoc@aol.com
Shepp & Associates LLC	Andrew Shepp	1312 Washington Avenue, 6th Floor St. Louis MO 63103, USA Tel: +1 (314) 621-3272 Fax: +1 (314) 621-4267 Email: ashepp@compuserve.com
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@interlog.com
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	Benoit Paquin	Soliton Denmark, Havsgårdsvej 4, 2900 Hellerup, Denmark Email: sales@soliton.com
SovAPL Russian Chapter of SIGAPL	Alexander Skomorokhov	PO Box 5061, Obninsk-5, Kaluga Region 249020, Russia Tel: +7(08439)47109 Fax: +1 (530) 6885510 Email:askom@obninsk.com
Strand Software Inc	Anne Faust	P.O. Box 330, Excelsior, MN 55331 USA Tel: +1 (952) 470-7345 Fax: +1 (952) 470-9202 Email: info@strandsoft.com
Rex Swain	Rex Swain	8 South Street, Washington, CT 06793 USA. Tel: +1 (860) 868-0131 Fax: +1 (860) 868-9970 Email: rex@rexswain.com
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
Sykes Systems Inc	Roy Sykes Jr	4649 Willens Ave., Woodland Hills, CA 91364, USA Tel: +1-818-347-5779 Email: roysykes@earthlink.net
Toronto SIG	Richard Procter	PO Box 55, Adelaide St. Post Office, Toronto Ontario M5C 2H8, Canada Email: info@torontoapl.org
Weighahead Systems	Philip Bulmer	Camberley House, 1 Portesbery Road, Camberley, GU15 3RB, UK. Tel +44 1276 20789 Email: sales@weighahead.com
Stephen Wynn		8 Clarence Gardens, Brighton, Sussex BN1 2EG, UK. Tel: 01273-327238 Email: centre@boltblue.com
Zark Incorporated	Gary A. Bergquist	23 Ketchbrook Lane, Ellington CT 06029, USA. Tel: +1 (860) 872-7806

FTP SITES

IBM APL2	ftp.software.ibm.com/ps/products/apl2
Waterloo Archive	archive.uwaterloo.ca/ftparch/languages/apl
APL-to-ASCII	archive.uwaterloo.ca/languages/apl/workspaces/aplascii

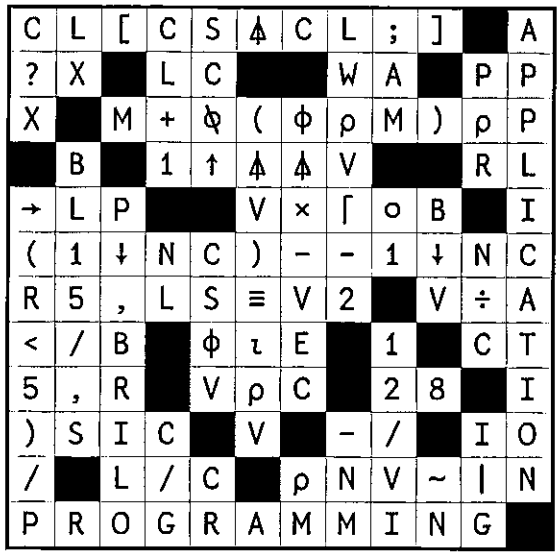
WORLD WIDE WEB SITES

ACM SigAPL	www.acm.org/sigapl/
Adapta Software	www.adapta.de/
Adaptable Systems	www.assuredsystems.com.au/
AFAPL	www.afapl.asso.fr/ (Journal available on line)
APL2000	www.APL2000.com/
APL-385	www.demon.co.uk/apl385/
APL Forum	www.aplforum.com
APL Journal, Germany	www.rhombos.de/apljourn.htm
APL Borealis Inc.	www.aplborealis.com

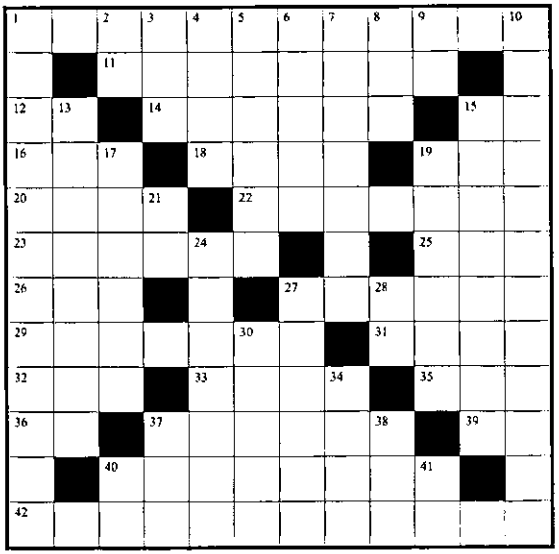
APL Systems IDC SL	www.apl2000.net
AUSCAN	http://home.ca.inter.net/rjp/auscan/
Eke van Batenburg	wwwbio.LeidenUniv.nl/~Batenburg/index.html
Carlisle Group	http://carlislegroup.com/
Causeway	www.causeway.co.uk/
CODEWORK	www.codework-it.com/tangram/eng/
CoSy (Bob Armstrong)	CoSy.com/
Dinosoft Oy	www.dinosoft.fi/
Dittrich & Partner	www.dpc.de; www.apl-online.de
DMOZ - Open Directory	http://dmoz.org/Computers/Programming/Languages/APL/
Dyalog Ltd	www.dyalog.com/
DynArray	www.dynarray.com/
FinnAPL	www.pyr.fi/apl/
Godin London Inc	www.godin.com/
Houben	http://home.wanadoo.nl/houb/IDA.htm
Hoekstra Systems	www.HoekstraSystems.ltd.uk/
IBM APL2	www.ibm.com/software/awdtools/apl
Infostroy	www.infostroy.ru
Insight Systems ApS	www.insight.dk/
Japan APL Association	www.naska.co.jp/JAPLA/
JSoftware Inc	www.jsoftware.com/
KJK-tieto Oy	www.kjk-tieto.com
kx systems	www.kx.com
Lambent Technology	www.lambenttechnology.com
Lescasse Consulting	www.lescasse.com/
Lingo Allegro USA Inc	www.lingo.com/
Mackay Kinloch	http://mackaykinloch.ltd.uk/
MicroAPL Ltd	www.microapl.co.uk/apl
Milinta Inc	www.milinta.com
Oasis b.v.	www.oasis.nl/
Optima Systems Ltd	www.optima-systems.co.uk
Qualedi, Inc.	www.qualedi.com
Renaissance Data	www.aplbooks.com/
Resources & Results	www.decision-support.net
RE Time Tracker Oy	www.rett.fi/
Right Seat Software, Inc.	www.voxproxy.com
The Rochester Group Inc.	www.rochgrp.com/
Shepp & Associates	www.digitravel.com/
SigAPL	www.acm.org/sigapl/
Soliton	www.soliton.com/
Snake Island Research Inc.	www.snakeisland.com
Strand Software Inc.	www.strandsoft.com/
Rex Swain	www.rexswain.com/
Toronto SIG (for Toolkit)	www.torontoapl.org/
Jim Weigang	www.chilton.com/~jimw/
Weighahead Systems	www.weighahead.com

The ZARK Crosswords
20.3 Solution and New Crossword

Solution to Crossword in 20.3



New Crossword



Across

1. APL, to the confused Greek novice
11. 1|PIF
12. □___ to pause awhile
14. (⌽C)≥T without using >, ≥, <, ≤
15. □___ to show a function as a character matrix
16. 2|N tells whether integer A is ___
18. (+/ER)⍴2 for positive integer vector ER
19. (-M)+M=M
20. Is N at least as big as e raised to the R power?
22. The magnitude of the sum of V and each of the first five elements of BU
23. *(⊛ED)-(⊛LTH), the easy way
25. Rank (number of dimensions) of T
26. 'FLAPPABLE' ~ 'APL'
27. Ratio of M to A, displayed to BR places
29. B random numbers from OC-VC to OC-1
31. Area of rectangle GA wide by T long
32. ''⍴(R/⍳⍴R),□IO+⍴R for bit vector R
33. (×/⍴T)⍴T+2

35. (0,I)[⎕IO+T] for numeric scalar I, bit vector T
36. ⎕__ returns the available characters
37. round the elements of M to the nearest integer
39. "→" ~ 'TO'
40. Wρ-S×Y for scalars W, S, Y
42. ⍞ uses the method of _____

Down

1. ^/4 1 ...
2. ⎕__, number of significant digits to display
3. Keyboard abuse: "___ Enter"
4. Flag elements of matrix AF matching scalar E
5. Remainder of H÷TR subtracted from B
6. Smallest number in vector V or scalar E
7. Two more than vector KR. rotated TP to the left
8.)____ is the same as)RESET
9. APL has consistent order __ execution
10. ⍋V and ⍋⍋V are _____ of ⍳ρV
13. VV∈DBC for 1=ρρDVC, LD,ρDBC, ⎕IO←1
15. Turn the reciprocal of the cumulative product of G into a matrix with shape B and subtract it from C.
17. D raised to the power 1÷(¯1+E)
19. Number of rows in MAT
21. ⎕__ is the "seed" for generating random numbers
24. Random integer from T+1 to 20, ⎕IO=1
27. BC,[2]Q[;5/⍳1↓ρQ]
28. Character representation of numeric G
30. +/V×S, V a vector, S a scalar
34. +/T,SU for scalars T and SU
37. Smallest number in vector S
38. M,[2]((1↑ρM),1)ρA for matrix M, scalar A
30. ⎕__ is the space remaining
41. SC, MN, HR, DY, WK, MN, __,DC, CN, ML

J-ottings 40: Parallel Joins

by Norman Thomson

J programming for data parallelism (as opposed to processor parallelism) might well be recognised in other languages as a design pattern. To put this in another way, upgrading verbs developed for simple data so that they also work parallel-wise on data with more complicated list structures is a generic problem. In the case of scalar verbs there is often nothing to do, for example 1 2 3 + 4 5 6 carries out additions in parallel, and 2 + 3 4 5 extends 2 to 2 2 2 prior to doing the parallel additions. It is often desirable to upgrade more complicated verbs so that they behave like scalar verbs. There are three mechanisms available for achieving such parallelism in J, explicit acknowledgement of which may help avert the untidiness of experimenting more or less at random, until eventually the desired result emerges (or in some cases, not!). Those who have never had this heuristic experience in J are probably highly skilled programmers who have no need to read any further!

The first mechanism for parallelism is **rank control**. Start with append as an example, which has a minimum operating rank of 1:

```
   1 2,3 4 5          NB. list joined to list
1 2 3 4 5
```

Provided that the appending is to take place at equal rank levels of the arguments, it does not matter what argument is given to an explicit rank conjunction provided that it is a strictly positive integer:

```
   1 2,"(17)3 4 5     NB. same as 1 2,3 4 5
1 2 3 4 5
```

Rank control is about operating at different levels of the left and right arguments, for example 'scalar to list' or 'scalar to list of lists' :

```
   1 2,"(0 1)3 4 5          NB. join scalars to a list
1 3 4 5
2 3 4 5
   1 2,"(1 0)3 4 5          NB. join a list to scalars
1 2 3
1 2 4
1 2 5
```

Sometimes an operation cannot be performed without **length equalisation** using a **fill character** as in

```
   4 5 6 7,"(1 2)i.2 2  NB. join a list to a list of lists
4 5 6 7
0 1 0 0
2 3 0 0
```

Length equalisation and fill characters are *provisions* rather than controls, and are present because the language designers wisely judged that they are greatly preferable to length errors.

Parallelism for joining requires adjusting append so that the upgraded version automatically

(a) joins several lists to the same list of scalars;

(b) joins the same list to several lists of scalars; and

(c) joins matched pairs of boxed lists.

Achieving this requires **scalarisation**, which is the second major control. It is tempting to equate scalarisation with 'box', but this is not quite accurate since link (;) also performs scalarisation. The results of scalarisation are non-simple scalars, which cannot take part in, for example, basic arithmetic operations, even although their *contents* can. Thus any operations on contents require an open, and in the words of the help file "opened atoms are brought to a common shape". To put this in another way, open always forces **rectangularity** which is sometimes what is wanted, at other times not. The only operations in which non-simple scalars can take part in are structural ones such as joining, shifting, rotating and transposing.

To return to the requirements above, consider (a). `1 2;1 3` is a list of two boxed non-simple scalars which must be opened before each is joined to a simple list such as `7 8 9`. Following the remarks in the previous paragraph, no joining can take place until the two boxed lists are opened, whereupon the result is 'list joined to list', that is append at rank 1:

```
   (>1 2;1 3),"(1)7 8 9          NB. join each of two lists to a
list
1 2 7 8 9
1 3 7 8 9
```

There is a snag, though, namely that if the two lists on the left are of unequal length the rectangularity rules will enforce an irrelevant fill character sitting astride the join :

```
   (>1 2;4 5 6),"(1)7 8 9   NB. join each of two lists to a list
1 2 0 7 8 9
4 5 6 7 8 9
```

The fill character can be pushed to the right by scalarising the right hand side, and bonding the append with open:

```
   (1 2;4 5 6),&><7 8 9          NB. join each of two lists to a
list
1 2 7 8 9 0
4 5 6 7 8 9
```

The success of the above 'append each' operation depends on **scalar expansion** of the right argument, following which the compound verb , &> behaves like a scalar verb such as + or * only with a fill character. If the latter is not acceptable then use compose (&.) rather than bond and work with boxed lists:

```
   (1 2;4 5 6),&.><7 8 9    NB. join each of two lists to a list
+---------+-----------+
|1 2 7 8 9|4 5 6 7 8 9|
+---------+-----------+
```

This underlines the fact that u&.v (compose) is more subtle than 'do v, apply u, undo v' (in this instance 'open, append, box'), because if this were so, there would be an embedded fill character present in the result.

For requirement (b), suppose the list 1 2 has to be joined to the separate lists 4 5 and 7 8 9. The foregoing discussion leads to

```
   (<1 2),&.>4 5;7 8 9      NB.join one list to each of two lists
+-------+---------+
|1 2 4 5|1 2 7 8 9|
+-------+---------+
```

Finally requirement (c), which, perhaps surprisingly, turns out to be the simplest of the three because there is no need for scalar expansion to be invoked by an explicit box :

```
   (1 2;2 4 6),&.>4 5;7 8 9 NB. two lists joined pair-wise
+-------+-----------+
|1 2 4 5|2 4 6 7 8 9|
+-------+-----------+
```

So to start the quest for a parallel joining verb, define

```
   je=.,&.>            NB. join each without fill characters
(1 2;4 5 6)je<7 8 9
```

so that the three requirements (a), (b) and (c) are dealt with by

```
   (1 2;4 5 6) je<7 8 9      NB. join many to one
   (<1 2) je 4 5;7 8 9       NB. join one to many
   (1 2;2 4 6) je 4 5;7 8 9  NB. join lists pair-wise
```

respectively, and the basic case, that is using je to append and box two unscalarised lists, is:

```
   1 2 (je&<) 3 4 5          NB. box following append
+---------+
|1 2 3 4 5|
+---------+
```

Thus je can deal with all possible cases, but only if the responsibility for boxing unboxed arguments is borne by the user. To automate this decision use:

```
   unboxed=.-:>          NB. 0=boxed, 1=unboxed
   box=.]`< @.unboxed    NB. box if unboxed, else do nothing
```

so given

```
   JE=.(,&.>)&box        NB. je after boxing any unboxed arguments
```

cases (a) to (c) and the simple case are given by

```
1 2 JE 3 4 5               NB. two lists, append and box
(1 2;4 5 6) JE 7 8 9       NB. cf. 1 2 + 3
1 2 JE 4 5;7 8 9           NB. cf. 1 + 2 3
(1 2;2 4 6) JE 4 5;7 8 9   NB. cf. 1 2 + 3 4
```

With character strings it is natural to think in terms of suffixing and prefixing in the obvious literary sense. It is instructive to see why some plausible ideas for adding different inflections to the same verb root fail. First try stitching:

```
   (<'post') ,.'s';'ing';'ed'
+----+---+
|post|s  |
+----+---+
|post|ing|
+----+---+
|post|ed |
+----+---+
```

This goes part of the way but there appear to be fill characters present even in the absence of an explicit open.

However, the ravel of the above expression

```
   ,(<'post') ,.'s';'ing';'ed'
+----+-+----+---+----+--+
|post|s|post|ing|post|ed|
+----+-+----+---+----+--+
```

shows that the fill characters are purely for the exigencies of display, although the division into three separate words has been lost. Replacing ravel with raze is even worse:

```
   ;(<'post'),.'s';'ing';'ed'
postspostingposted
```

Using rank control removes boxes and preserves the words as entities but at the expense of introducing genuine fill characters:

```
   ;"1(<'post'),"1 0 's';'ing';'ed'
posts
posting
posted
```

But do not worry - help is at hand through JE :

```
   'post' JE 'ing'
+-------+
|posting|
+-------+
   'post' JE 's';'ing';'ed'
+-----+-------+------+
|posts|posting|posted|
+-----+-------+------+
   ('post';'eat';'flower')JE 'ing'
+-------+------+---------+
|posting|eating|flowering|
+-------+------+---------+
```

What happens if JE is applied with explicit rank control? First, qualification with zero rank, that is JE"(0), makes no difference since the items in the left and right arguments are treated either as scalars or as lists of scalars anyway.

The interesting cases are therefore :

```
   ,(1 2;2 4 6)JE"(1 0) 4 5;7 8 9          NB. all join
combinations
+-------+---------+---------+-----------+
|1 2 4 5|2 4 6 4 5|1 2 7 8 9|2 4 6 7 8 9|
+-------+---------+---------+-----------+
   ,(1 2;2 4 6)JE"(0 1) 4 5;7 8 9          NB. all join
combinations
+-------+---------+---------+-----------+
|1 2 4 5|1 2 7 8 9|2 4 6 4 5|2 4 6 7 8 9|
+-------+---------+---------+-----------+
```

which, using a ravel on the left in both cases, give lists of all possible joins, but in different orders.

Also the effect of

```
   ,1 2 JE"(0 1) 4 5;7 8 9            NB. all join combinations
+-----+-------+-----+-------+
|1 4 5|1 7 8 9|2 4 5|2 7 8 9|
+-----+-------+-----+-------+
```

is to scalarise the individual items of the left argument which now behave as two one-item lists.

The third mechanism for parallelism is that of the **join alternatives**, namely stitch and laminate. Stitch (which used to have the more descriptive name of "append items") provides an alternative for rank control, but has more restrictions due to constraints of length compatability.

```
   1 2,"(0)3 4              1 2,.3 4
1 3                     1 3
2 4                     2 4
```

Laminate joins by introducing a new 2-list which means that it is less easy to find simple direct alternatives to append. Also, unlike stitch, length equalisation and fill characters can be tolerated, as illustrated in the example below:

```
   >1 2 ,&box 3 4 5            1 2 ,: 3 4 5
1 2 0                     1 2 0
3 4 5                     3 4 5
```

The verbs `JES` and `JEL` are identical to `JE` except that append has been replaced by stitch and laminate respectively. It is helpful to see the differences by comparing results side by side The boxes now contain lists of lists rather than

lists; in loose terminology, stitch adds things on the right, laminate adds them on the bottom.

```
   1 2 JES 3 4              1 2 JEL 3 4
+---+                    +---+
|1 3|                    |1 2|
|2 4|                    |3 4|
+---+                    +---+
   1 2 JES 3 4;5 6              1 2 JEL 3 4;5 6
+---+---+                +---+---+
|1 3|1 5|                |1 2|1 2|
|2 4|2 6|                |3 4|5 6|
+---+---+                +---+---+
   (1 2;3 4)JES 5 6         (1 2;3 4)JEL 5 6
+---+---+                +---+---+
|1 5|3 5|                |1 2|3 4|
|2 6|4 6|                |5 6|5 6|
+---+---+                +---+---+
   (1 2;3 4)JES 6 7;8 9          (1 2;3 4)JEL 6 7;8 9
+---+---+                +---+---+
|1 6|3 8|                |1 2|3 4|
|2 7|4 9|                |5 6|7 8|
+---+---+                +---+---+
```

In summary, rank control, scalarisation and stitch/laminate are means of attaining parallel joins, and in combination give a considerable range of possibilities. Of course the value of such techniques is in the interesting things which can be done with them - but that must wait until later!

RECENT MEETINGS

In this issue, we cover .....

- The BAA's 20th Anniversary K day at the Royal Society
- The APL Moot at Great Hallingbury
- The 8th Annual FinnAPL Forest Seminar
- The APL Germany Spring Meeting at Bingen

Thanks to everyone for the notes and pictures.

BAA 20th Anniversary K day

reported by Adrian Smith (adrian@causeway.co.uk)

Acknowledgements

Arthur Whitney (and his considerable entourage) all worked really hard to make this a success. The Royal Society provided the venue, and followed it up with an excellent meal afterwards (in slightly premature celebration of 20 years of Vector). Everyone who came had a good time, and felt that we should do more of this sort of thing. Ray promptly organised an APL Moot in a shed near Stansted, and that went off well too. It is reported next.

The KX Users Group Meeting

Stephen Taylor & I were privileged to be invited along to the *KX User's Meeting* which occupied the whole day before the BAA workshop. The morning was taken up by a superb exposition of K4 (the latest incarnation of the K language) and attendees were provided with a temporary copy to try out during the meeting. It expired exactly 24 hours later, which gives you an idea how protective KX are feeling about their new baby.

The afternoon was occupied with a series of user-presentations. All I can say about these was that the content was seriously impressive, and that the volume of data routinely handled by Kdb (current release) can leave an audience stunned. Any more, and the corporate lawyers would be after us, so back to the relatively safe ground of K4 and Arthur Whitney.

Arthur on K, Kdb, K4 and Kdb+

My notes are a mixed bag, taken initially from Arthur's talk to the Users Group, and consolidated with material gleaned from the BAA workshop the day after. Here are a few opening quotes:

> "The best programming language is the one where the program is the shortest"
>
> "The only programs that have a chance of being correct are the short programs"
>
> "Words instead of symbols is fine – orthographic terseness is a bonus"
>
> "If you find something in the system that could be shorter – let me know!"

So the goal of K4 and Kdb+ is very simple – to be the world's best programming language coupled to the world's best database. From the language side we have concepts like 'Atom' and 'List' and 'Dictionary' and from the database we have 'Select' and 'Delete', 'Update', Insert' and 'Join'. How to put them together in a way which makes sense?

From A+ to K to K4

Arthur's attitude to A+ fascinates me – it was a massive success at Morgan Stanley " ... we did everything in A+. Trading systems, LP, computer-aided instruction, fixed income etc. All in A+ ..." and yet " ... I didn't like A+ 'cos its a lousy language!". Which is very interesting, as I still think A+ is one of the best APLs around. However I also like quirky buildings with bits missing (like Ely cathedral) in preference to anything mathematically pure and technically 'correct' so maybe I just enjoy A+ because it's a mass of pragmatism and compromise?

Anyway, onward from K to K4, and some ideas where Arthur is headed next.

His first objective (as always) is speed, which is why the immensely successful Kdb is being redone as native C code, rather than its current implementation as a library of K functions. This I can immediately relate to, and I am certain it will be an instant winner with his Kdb user-base. I am less clear about the migration to K4 particularly as "it is about 90% compatible with K3, but that is enough to kill any script". Some of the key features are:

- the language has about 40 datatypes, 40 monads and 40 dyads, so roughly 1600 × 40 method signatures. Probably a few thousand are commonly used. The sourcecode is currently 9k bytes, so the average method length is around ½ a byte (K gets smaller with every build, but this has to be a record).

- K4 is either 32-bit or 64-bit, and includes TCP/IP support (native) and a nice HTTP server as a script. Arthur showed some of the K4 source for this, but may have wished he hadn't - it takes terseness to new extremes and at least some of the audience were old-fashioned enough to appreciate the value of white-space in programs!

- K4 is headed to having tables as native data-types. This requires raw language support for many more data-types (e.g. 000011b is a boolean vector) including .Net types like DateTime. In fact the K4 list of types maps almost exactly to the .Net Common Language Runtime, with some extra little structures added.

The move to 64-bit addressing has taken much longer than anyone expected. "In 1992 we thought we couldn't live another day without 64bit. Now (eventually) people are hitting WS FULL so the move to 64bit is going to happen." The target hardware has 32-64 Gbytes of main memory, as mapped files become the bottleneck otherwise. Arthur put the addressibility limit at about 2 × physical memory. "Better to get WS FULL than hit the pagefile".

Dictionaries and Tables

This (assuming it all works) is a major foray for functional programming into the world of relational tables. It was not clear from the presentation how much of this is still experimental, but there are clearly lots of edge-effects that can really spoil your day when you turn loose the APL primitives into a database.

Health-warning - the syntax could change and my notes were sparse, so take this as a flavour, not as a commitment from anyone to do it exactly like this!

```
d: `a `b  ! 2 3
   domain ! range
```

This gives us an array which is addressible by key (nothing new about that) but what is new in K4 is that we can do arithmetic with it:

```
d + d
`a `b ! 4 6
```

So far, so obvious. Now let's make another one:

```
e: `a `c ! 3 4
```

The challenge is how to 'add' two relations where the keys don't match up.

```
d + e
`a `b `c ! 5 3 4
d - e
`a `b `c ! -1 3 -4
```

"This stuff is kind of neat ..." and implements at the primitive level stuff which many of us have spent many happy hours coding around. Usually at the point where databases and APL meet up. What bothers me is that Arthur has opened so many new Pandora's boxes here in one go that he may never quite achieve the simple, logical consistency he craves. For example:

```
d & e
`a `b `c ! 2 3 4      min is obvious
d | e
`a `b `c ! 3 3 4      max is obvious
d,e                   catenation needs a bit of thought
```

As soon as you hit catenation, you are forced to choose between the 'set' paradigm, the 'multiset' paradigm and the 'list'.

- a **set** like (2 3) is a simple unordered collection of unique values
- a **multiset** allows repeats but does not care about ordering (2 2 3 = 2 3 2)
- a **list** allows repeats and preserves ordering.

Sets are useless for arithmetic, multisets are useless for answering questions like 'where did it change?' so there are plenty of good answers to this one. Arthur's choice was to treat it as an override "try e - if e fails use d" which I would never have guessed at as a possibility!

What about:

```
d[`b `b `a `c]
3 3 2 •
```

If you allow indexing by keys, you need a null to represent the 'undefined' state. OK, suddenly we have nulls so it gets tempting to allow:

```
1 2 3 + 1 2
2 4 •
```

... and so it goes. If you allow dictionaries on both axes, you end up with a cross-tab, and (of course) Arthur has made sure the arithmetic primitives all work on cross-tabs too.

Summary

Kdb+ re-implements Kdb and offers speed and almost infinite memory. It looks to have a ready market, and it is hard to see how it can fail. K4 takes K to a new level of abstraction and technical brilliance. I just hope Arthur can finish it!

Report from the APL-moot

notes by Stephen Taylor

Twenty-one Vector programmers met in the Village Hall in Great Hallingbury for the first APL-moot.

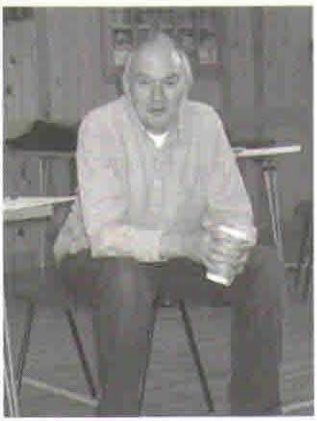
We brought computers, frisbees, network hubs, wine, an espresso machine and a bread maker, a data projector, sleeping bags and the first series of Farscape.

Ray Cannon produced greasy fried breakfasts every morning; Stephen Taylor, life-endangering coffee; and Optima Systems treated us to wine and pizza on the Friday.

Paul Mansour with Veli-Matti and Morten

We limited the plenary sessions to provide time for 'birds of a feather' huddles. If you weren't there, this is what you missed!

- Paul Mansour demonstrated fast GUI development
- Morten Kromberg showed us VectorLib
- Ray Cannon helped us install and review his User Commands
- Brooke Allen had us playing his Behavioral Finance Game on our network
- John Scholes puzzled with Euler over the bridges of Königsberg
- Richard Smith left his studies at Cambridge to show us his languages R and Rowan, developed using APL concepts and .NET components




John Scholes and Richard Smith (relaxing)

Most people had brought either software and ideas to share, or questions to discuss. So we spent a good deal of our time in (poster-free) birds-of-a-feather sessions:

- Paul Mansour: namespace programming techniques and documenting OOP objects
- Morten Kromberg: Embedded Terminal Services
- Christian Langreiter: the K language and numeric arrays
- Michael Baas: Dyalog & Outlook and Adapta DLS
- Ray Cannon: Pocket APL and Quad-NA calls

- Veli-Matti: processing very large (6Gb) text files
- Eke von Batenburg: control structures; porting between APL2000 and APLX; and APL2C
- Graeme Robertson demonstrated his array explorer
- Peter-Michael Hager: cryptography and anonymous objects
- Stephen Taylor: Mailtrain (spooling Word documents from APL) and functional programming style
- Ian Clark: developing online APL tutorials

Dick Bowman and John Daintree of Dyalog joined us for Saturday. Paul Grosvenor managed us through a rapid clean up, and we finished with a long Sunday lunch at the village pub.

What worked, what didn't

The concept of the moot worked very well, with people flying in from Germany, Austria, Finland, Denmark, Holland and the United States. The hall suited us, though the rainy hike to the pub didn't. Stansted was conveniently if noisily close, but most flyers travelled via Heathrow!

We appreciated the ample time available for us to work in small groups, and agreed we might skew the balance of time even further in future. Paul Mansour and Brooke are contemplating a moot in the eastern United States; Ray another one in late summer.

FinnAPL Forest Seminar VIII

reported by Jouko Kangasniemi

The 8th annual Finnish Forest APL seminar took place on 18.3.2004-19.3.2004 at the Vanajanlinna castle. The distance from Helsinki is almost 100 km. There were 35 participants this year. The guest speaker was David Liebtag, one of the main developers of IBM APL2.

The seminar was held at the Ratsula-house (stables in English), which is 500 metres from the main building. All the hotel rooms close to the venue were occupied by the participants, so we got our own APL-privacy without any disturbances by the (real) outside world. These are my notes of the event.

Olli Palosaari, Eija Ahopelto, Kimmo Kekäläinen and David Liebtag get close to water

The seminar was started at 09:15 by Antti Korpela, the president of the Finnish APL association. The welcome coffee was a bit late, but we all got it nearly in time. Veli-Matti Jantunen from Statistics Finland was the first speaker. He educated the crowd in how to handle (extremely) big text files with Dyalog APL. The presentation was informative and colourful, as always. Personally I have not even dreamt of handling files whose size can be over 10 megabytes, but the treatment was relatively easy with VM's tools. Veli-Matti's current challenge is to tackle gigabyte-size files.

Olli Palosaari shed light on the APL-history of the Employers' Confederation of Service Industries from the 1980s to today. The mainframe has been replaced with workstation Dyalog APL within that time. Wage statistics, business surveys and member registers are being treated with APL tools.

David Liebtag presented the wonders of modern APL2 in the Windows environment. Mainframe APL is familiar to many of us, but today's APL2 was distant to most of the audience. The modern windows development system seems to be as good as the others. Several auxiliary processors and good connections to other programming tools gave the impression, that I could solve my programming problems with APL2 also.

Lunch time. Yam! After lunch was the presentation of the history of the Vanajanlinna castle. It was originally built as a hunting cottage nearly 100 years ago for a well off Finn called Carl Wilhelm Rosenlew. Then it was owned by a German industrialist for a short period of time. After the second world war it was a training facility for communists. It went bankrupt quickly after the Soviet Union collapsed, and now the castle is owned by the city of Hämeenlinna. It is a very popular place for meetings and seminars, and they are now building a golf course which makes it even more popular.

Göran Koreneff from VTT Technical Research Centre of Finland introduced the wonders of electricity markets, the prices and the volumes. He has built an application that can model and handle data even by the hour. Dyalog APL and Causeway tools were used. Lots of Rainpro-graphs.

Reima Naumanen showed how they have boosted the Dyalog APL session at MT-palvelut. The first thing to do when starting the development environment is to load some tools to session. There Reima then has the utilities for small and even bigger tasks. Workspace comparison and tracking of changes was swift. And with added namespaces and the new stuff that comes with version 10, one must conclude that Dyalog APL can be a very effective development environment. Reima mentioned that he uses an additional monitor simultaneously with his laptop, which I also had to try at home, and now I have double as much screen size as before.

Vanajanlinna's seminar rooms have all wireless data networks, but Reima's 30 metre long data cable made it unnecessary. You could read your email in every corner of the room with that, and it would have been possible also in the car park.

Markku Koskela from Dinosoft presented how to make dynamic web graphics with dotnet-tools. Dinosoft has managed to combine Causeway's graphic tools with a statistics web tool called *PxWeb*. The application produced quality graphs

swiftly. The engineering involved was a bit complex to me, because not only Dyalog APL but also Javascript and C# were used. We must return to this in further meetings.

Then was the time for the traditional sauna-seminar. And after that dinner. Then we went to the wine cellar, where we had some refreshments. Also singing and vivid conversation took place until it was (well past) time to go to bed.

Friday morning was the first real spring day, and the sun was shining brightly as ever. I dragged myself at 09:00 to the seminar room and showed how to calculate the probability of a downturn. There Dyalog APL churns some Markov processes and Bayesian probabilities.

Antero Ranne from Ilmarinen Mutual Pension Insurance Company presented the investment optimization model of a pension fund. The aim is to find the best portfolio with the desired risk level. Bankruptcy is something one tries to avoid. Dyalog APL is used as a part of a large application. We saw some nice real time graphics at the time when calculation proceeded.

Ismo Risku and Peter Biström from the Finnish Centre for Pensions introduced us to new long term forecast model for pension expenditure. One could model the development of expenditure for years and years and... We also saw a population forecast model done with Dyalog APL.

Hannu Virtanen from Tietoenator showed the web services interface to Eporttiportal. Hannu claimed that the technique is simple and "low tech". Somehow I am not going to try it right away, because I still have some dark spots in my XML, SOAP, TCP/IP etc. knowledge. Good to know that all that can be done with Dyalog tools.

Then it was David again. He and his colleagues had been lately spending their time with a Java interface of APL2, and we got the honour to be one of the first ones to see it. At the beginning of his presentation David promised to make us all Java programmers in 5 minutes. I do not know if it is the teacher or the student to blame, but in my case I still need at least some extra minutes of education before starting my Java business. But anyhow, APL2 can talk Java and vice versa.

Then it was time for Antti Korpela's closing remarks, which ended the seminar. The fellows from Tietoenator transported me all the way to my home stairs, which was very comfortable. A good seminar once again, and maybe I'll be there again next year.

APL Germany, Bingen, April 30th 2004

reported by Adrian Smith (adrian@causeway.co.uk)

Acknowledgement

Thanks again to Dieter Lattermann and the rest of the German group for the invitation. Bingen is a really easy trip from the UK (Frankfurt + 50 mins on the train gets you 100m from the hotel), so for once the travel was very simple. On the return leg, we were greeted at the departure gate by a very sheepish pilot - the tail skid on the 767 had stuck 'out' so they could fly OK, but could not discharge any water from the rear of the plane for fear of it freezing on. So we were all sent off to the bathroom before boarding!

The Meeting

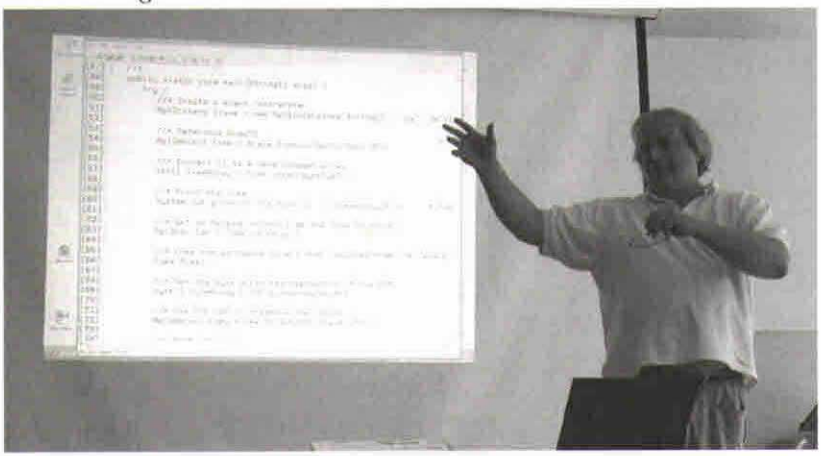
As usual, this was the second part of a 2-day meeting (day-1 was the formal GSE session with IBM) and opened with a short AGM before moving on to a good selection of presentations.

Dieter Lattermann opening the meeting

Around 60 people attended the 2 days, although a few did escape after the (most interesting) winery trip on the first evening. David Liebtag was the first main speaker, and summarised the new features in APL2 CSD4 which allow for a very

clean interface from APL2 to Java (and from Java to APL2). He has provided a very thorough introduction to the Java API (which you can read on page 101) but here is my impression of the key points.

Working with Java and APL2
by David Leibtag

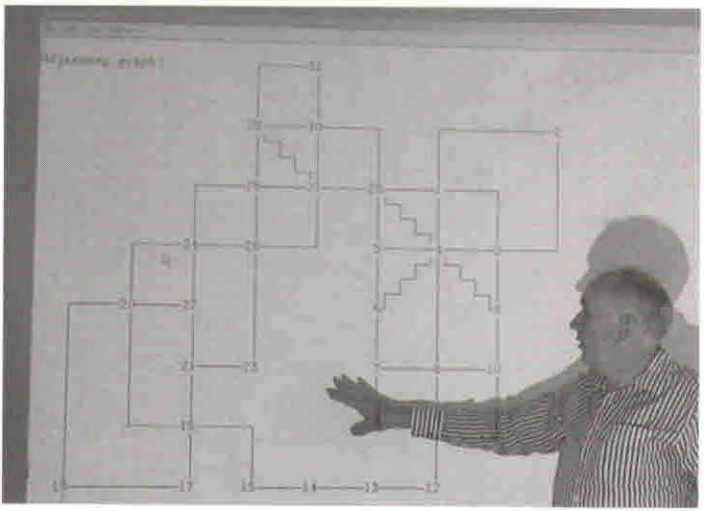


Here is David, and here is some Java code calling APL2. That more-or-less summarises what IBM have done – made APL2 callable easily from other programming environments. They have designed a specific API for Java and a general-purpose API for C which should be usable from any language. The APL that you connect to is very similar to the runtime environment, so there are no system commands, just raw code and user-defined functions. To make this readily shareable, the interpreter is now a very small .EXE file, with a much larger (shared) DLL portion to allow multiple instances to start quickly.

David also showed APL2 calling Java via a new Associated Processor AP14 which creates an instance of a Java class, and gives you back a handle, which allows ready access to any public methods or instance data. He emphasised the need for good housekeeping here – if you fail to free the instances your application will gently leak memory until)OFF recovers it for you. He pointed out that this allows you to use all the threading capabilities of Java to launch background tasks which run in asynchronously in independent threads, and exploit multi-processor architectures where available. It all looks very solid, very useful, and (probably the primary motivation) it allows APL2 applications to integrate seamlessly with IBM's mainstream *WebSphere* developments.

John Scholes **in Dogger and South Utsire**

Not his real title, but he made the mistake of using the shipping forecast in one of the examples. Basically, this talk was about graphs (the mathematical kind when you have vertices and edges) and here is John explaining the knotty problem of locating Britain given only the connections between the sea areas listed in the daily weather forecast:



It all turns out to be a matter of finding chord-free regions in the map – the latest code is available on the DFns page at *www.dyalog.com* if you want to know more.

Once you have your graph, you can choose many ways to represent it, typically a boolean matrix (if most points link to most other points) or a nested vector of connections. Again, John has provided the full text of his talk (see page 55) so I will stick to a few highlights here.

He showed us the little DFns application to find the most efficient route across London using the classic Tube-map represented as a network. This uses the highly iterative approach of the original algorithm, but in APL you can improve by a factor of 4 by pushing a 'wave-front' through the network in parallel. I would very much like to have a go at compiling this one – I suspect that the 'simplistic' APL would probably be a better option when generated to raw C#, as simple scalar code runs very quickly indeed when compiled. See next-but-one talk!

New possibilities in Dyalog Version-10
by Peter-Michael Hager

This was an interesting exploration of the potential of un-named objects in Dyalog APL, Version-10. Peter-Michael showed the benefits of working with un-named Gui objects, TCP sockets and so on.

In Dyalog-9 there were many problems with the implementation, and much of what he showed required V10 to work correctly. You can read his notes in full on page 98.

For now, readers will have to content themselves with this black-and-white rendition of his splendid outfit. It is well worth a trip to the Vector website to experience it in all its colourful glory.

Adrian Smith on translation into C#

Fortunately, most of the delegates had read the PDF of the presentation beforehand, so I could use nearly all of the available time in showing examples of original APL code and the kind of C# that one can generate from it. This work is very firmly rooted in Tilman Otto's APL2C generator, but is entirely new code to target the .Net platform quite specifically. Several people were concerned that we were being too narrow in targeting C#, but I am still fairly relaxed about this, as (a) the *Mono* project already has the .Net Framework running on many Unix builds, and (b) it would be fairly easy to target Java instead of C#.

Although the prime motivation behind this project is ease of deployment, the thing that most caught the audience's attention was the potential to speed up existing APL algorithms by up to 30 times (for very iterative APL) or at least a factor of 5 (for typical commercial APL). You can read my detailed notes on this project on page 113, but hopefully it sparked enough interest that someone is now thinking of sitting down to write a bare-bones APL engine for .Net that is designed from the beginning to be readily compiled.

GENERAL ARTICLES

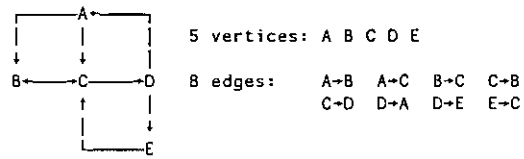
A Note on Graphs

from John Scholes, Dyalog Ltd (john@dyalog.com)

This is a short note on a large subject:

"A *graph* is a set of *vertices* (V), together with a set of connecting *edges* (E)."

In general, edges have a direction (V1...V2). Here is a simple graph:



We can represent graphs in a number of ways. Two of these are the *adjacency matrix* and the *adjacency vector*:

An adjacency matrix for the above graph would be:

```
  A B C D E
  ┌─────────      ⍝ Adjacency matrix for graph "a" above.
A │0 1 1 0 0
B │0 0 1 0 0
C │0 1 0 1 0    ⍝ Third row shows two edges: C→B  C→D.
D │1 0 0 0 1
E │0 0 1 0 0
```

... and an adjacency vector would be:

```
(2 3) (3) (2 4) (1 5) (3) ⍝ Adjacency vector for graph "a".
 A     B   C     D     E
           └─────────────  ⍝ Third item shows two edges: 3→(2 4): C→B C→D.
```

The above origin-1 vector can be changed to an origin-0 vector, merely by subtracting 1 from the nested array. All of the graph functions shown here are origin independent.

In general, the boolean adjacency matrix is better for dense graphs, where most vertices are connected to most others; and the adjacency vector is better for sparse graphs.

We can convert between the forms using:

```
{{⍵/⍳⍴⍵}⍤1↓⍵}     ⍝ vector from matrix form.

{⍵(⍳⍴⍵)∘.∈⍵}     ⍝ matrix from vector form.
```

Graphs are useful in a surprising number of applications. Examples include modelling transportation networks, job scheduling, resource allocation and determining the minimum number of postage stamps to stick on a letter!

Classical Breadth-First Search

We might want to explore the vertices of a graph in an orderly way. The classical breadth-first search algorithm maintains a *queue* of yet-to-be-visited vertices. Items are taken one at a time from the front of the queue, and vertices reachable from the head item are added to the back of the queue. The algorithm terminates when the queue is empty.

An APL coding of the classical algorithm might look like this:

```
bfs_c←{                          ⍝ Breadth-first search (classical).
    graph←⍺                      ⍝ graph.
    θ{                           ⍝ no vertices visited.
        ⍵≡θ:⍺                    ⍝ no vertices left: done.
        head←1↑⍵ ⋄ tail←1↓⍵      ⍝ next and remaining vertices in queue.
        next←head⊃graph          ⍝ vertices adjacent to head.
        (⍺,head)∇(tail∪next)~⍺   ⍝ accumulate visited vertices.
    }⍵                           ⍝ from starting vertex.
}
```

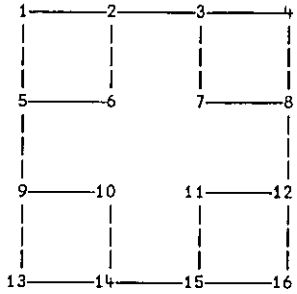
The inner tail-recursive function uses ⍺ as an accumulator for vertices already visited, and ⍵ as a queue for those still to be visited. By appending new vertices to the back of the queue and removing them from the front, we are assured of visiting nearer vertices before more remote ones.

Searching the simple graph from above, starting from vertex 5, yields:

```
      (2 3)(3)(2 4)(1 5)(3) bfs_c 5
5 3 2 4 1
```

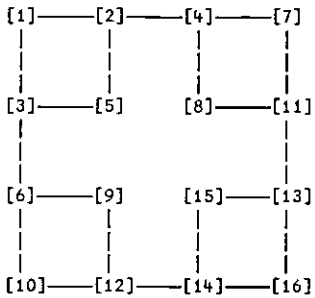
Using the following graph as a slightly larger example:

```
      graph←(2 5)(1 3 6)(2 4 7)(3 8)(1 6 9)(2 5)(3 8)(4 7 12),⍬
⍬:          (5 10 13)(9 14)(12 15)(8 11 16)(9 14)(10 13 15)(11 14 16)(12 15)
```



Graph Vertex numbers:

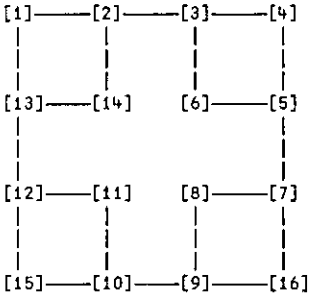
Breadth-first search from vertex 1 visits vertices in the following order:



BREADTH-first order of visits:

The function may be changed to produce a depth-first search, by *prefixing* new vertices to the queue, which then operates as a stack.

```
dfs_c←{                              ⍝ Depth-first search (classical).
    ...
        (⍺,head)∇(next∪tail)~⍺       ⍝ accumulate visited vertices.
    ...
}
```



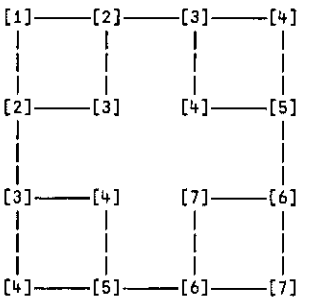

```
DEPTH-first order of visits:
```

Parallel Breath-First Search

In APL we can do a little better than the classical breadth-first search by processing ALL vertices at the same distance from the starting vertex in parallel. For non-trivial graphs, this leads to a significant performance improvement. Conceptually, starting from the originating vertext ⍵, the algorithm generates a "wave" (sometimes called a "fringe"), which ripples in all possible directions outwards through the graph; at each pass of the algorithm, the wave is advanced one step further from the starting vertex.

```
bfs←{                          ⍝ Parallel breadth-first search.
    graph←⍺                    ⍝ ⍺ is graph vector.
    ⍵{                         ⍝ from starting vertex.
        ⍵≡⍬:⍺                  ⍝ no unvisited vertices: done.
        adjv←graph[⍵]          ⍝ nested vector of ALL adjacent vertices.
        next←∪(↑,/adjv)~⍺      ⍝ simple vector of next unvisited vertices.
        (⍺,next)∇ next         ⍝ advance wave of visited vertices.
    }⍵                         ⍝ from starting vertex.
}
```

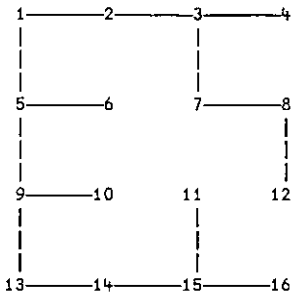
Using graph as above:




```
PARALLEL BREADTH-first order of visits:
```

Spanning Trees

When searching a graph, our routes from the starting vertex to each visited vertex form a "spanning tree". A breadth-first spanning tree for the above graph would look like this:




```
Breadth-first spanning tree from vertex [1]
```

```
span←{                  ⍝ Spanning tree for graph ⍺ from vertex(ices) ⍵.
    graph←⍺                   ⍝ ⍺ is graph vector.
    tree←¯2+(⍳⍴⍺)∈⍵           ⍝ initial spanning tree.
    free←{(¯2=tree[⍵])/⍵}     ⍝ free vertices in ⍵.
    {                         ⍝ graph is left operand.
        ⍵≡⍬:tree              ⍝ no vertices: done.
        next←free¨graph[⍵]    ⍝ next vertices to visit.
        back←↑,/⍵+0×next      ⍝ back links.
        wave←↑,/next          ⍝ wave front.
        tree[wave]←back       ⍝ set back links in tree
        ∇∘wave                ⍝ advance wave front.
    }⍵                        ⍝ from starting vertex.
}
```

We can use the spanning tree to find a shortest path to any vertex:

```
stpath←{                ⍝ Path through spanning tree ⍺ to vertex ⍵.
    tree←⍺              ⍝ (partial) spanning tree.
    ⍬{                  ⍝ path accumulator.
        ⍵<0:(⍵=¯2)↓⍺    ⍝ root or unvisited vertex: finished.
        (⍵,⍺)∇ ⍵⊃tree   ⍝ otherwise: prefix previous (parent) vertex.
    }⍵
}
```

... and combine the tree generation and path extraction into a single function:

```
      path←{                          ⍝ Shortest path from/to ⍵ in graph ⍺.
          graph←⍺                     ⍝ ⍺ is graph vector.
          fm to←⍵                     ⍝ starting and target vertex(ices).
          tree←¯2+(⍳⍴⍺)∊fm            ⍝ initial spanning tree.
          free←{(¯2=tree[⍵])/⍵}       ⍝ free vertices in ⍵.
          θ{                          ⍝ path accumulator.
              ⍵<0:(⍵=¯2)↓⍺            ⍝ root or unvisited vertex: finished.
              (⍵,⍺)∇ ⍵⊃tree           ⍝ otherwise: prefix previous (parent) vertex.
          }{                          ⍝ find partial spanning tree:
              ⍵≡θ:¯1                  ⍝ no vertices left: stitched.
              1∊to∊⍵:1↑⍵∩to           ⍝ found target: finished.
              next←free¨graph[⍵]      ⍝ next vertices to visit.
              back←↑,/⍵+0×next        ⍝ back links.
              wave←↑,/next            ⍝ vertex wave front.
              tree[wave]←back         ⍝ set back links in tree.
              ∇∇wave                  ⍝ advance wave front.
          }fm                         ⍝ from starting vertex.
      }

      graph path 1 11
1 5 9 13 14 15 11
```

Postage Stamps

An example of an application of graphs is to determine the smallest number of postage stamps to stick on a package. Suppose stamps are available in denominations of 1 5 6 10 26 39 43 and we need a total value of 51. One way to solve this problem is to repeatedly subtract the largest possible stamp from the remaining value:

```
   ┌──── remaining value
   │ ┌── stamps
  51 43
   8  5
   3  1
   2  1
   1  1
   0
```

... giving (43 5 1 1 1). However, on closer examination, (39 6 6) is a better solution.

We can solve this problem by making a graph whose vertices are the values 0, 1 ... up to the required postage, and whose edges connect each vertex to those vertices it is possible to reach by adding a stamp.

For the denominations above, the graph starts:
(1 5 6 10 26 39 43)(2 6 7 11 27 40 44)...

```
stamps←{                          ⍝ Postage stamps to the value of ⍵.
    ⍺←1 5 6 10 26 39 43           ⍝ Default UK stamp denominations.
    graph←⍺{⍵∘∩¨⍵+⊂⍺}⍳⍵+1         ⍝ values: 0 ⋯ ⍵.
    spath←graph path ⎕IO+0 ⍵      ⍝ shortest path 0→⍵.
    ¯2-/spath                     ⍝ best-fit stamps.
}
```

Then:

```
      display stamps¨51 52 53
┌→──────────────────────────────┐
│ ┌→─────┐ ┌→─────┐ ┌→─────┐   │
│ │39 6 6│ │26 26│ │43 10│   │
│ └──────┘ └──────┘ └──────┘   │
└∊─────────────────────────────┘
```

Further Reading

For more on Graphs, see: *www.dyalog.com/dfnsdws/n_Graphs.htm*

For a larger example, see: *www.dyalog.com/dfnsdws/tube_n_index.htm*

(For the moment, you will need to use IE6 as a browser in order to see the font correctly).

TABULA: a Scientific Calculator for the 21st Century

by Ian Clark (earthspot2000@hotmail.com)

Introduction

In 1972 Sir Clive Sinclair introduced the world's first pocket calculator, the *Sinclair Executive* - the size of a packet of 20 cigarettes [1]. Since then, pocket calculators have never looked back, though it has tended to be more of the same - more buttons, more functions, more memory, more battery performance. Is it possible to imagine a radical improvement to the original concept, even extending its provenance to desktop (or palmtop) computers?

Well... yes, easily. Long considered the techie's "Jedi sword", there's a need for a scientific calculator for the rest of us. As scientific literacy grows - as it must, if intelligent life is to continue (commence?) being found on planet Earth - the "intelligent laymen", someone like myself, whose scientific calculating skills have not improved at all since their zenith at high school (and a not-frightfully-good zenith at that!) has need of computational resources in the scientific review and decision-making area to match those already offered in the personal finance area by such products as TurboTax. In other words - every wannabe politician, journalist or planner needs one, every SF writer and eco-activist. And for much the same reasons too - so as not to leave all the accounting for the earth's resources in the hands of government or corporate wraiths.

The web runneth over with published statistics. A look at the *Seafriends* (New Zealand) website [2] shows a provocative blueprint for the numerate environmentalist's toolkit, a single web-page of statistics covering such basics as the metric SI system; densities (water, air, earth's crust); speed of sound, decay, absorption, attenuation; biomass conversions; radioactivity; human dimensions and needs, both individual and populations; energy costs and statistics; thermodynamics; chemical heats of formation, etc, etc. The trouble is: to apply all that concentrated data effectively you need to be a computational wizard, quite at home with applying formulas and units conversions, not to mention knowing what underflow and overflow is without having to call in the plumber.

Now, specialist calculators are available on the web for every purpose from Julian date conversion to pig-farming. For example, see: [3]. But you need to be a Java-programmer as well as a domain specialist to understand exactly what they do, or

else they're black boxes, magic rings, silently executing the will of their masters, the wraiths.

Hence the present project. Not exactly a tool to be welcomed by techies, TABULA assists you and me to answer questions like these, by a cross between speculative problem-solving and creative play:

1. You exercise to burn off a pound of fat. How many calories is that? How long would it light a 100W light-bulb?

2. If a 10ft fall is life-threatening on earth, what's the corresponding height on the moon?

3. A space cruiser on a slingshot orbit between Earth and Mars consists of two pods of equal mass tethered by a stiff boom. How fast does a 1000ft boom have to rotate to simulate Earth gravity in the pods?

4. A 20lb frozen turkey hangs motionless in space with respect to the stars. The moon sweeps it up. How much kinetic energy is released on impact in tons of TNT?

5. You are in a ship and see another ship explode. Six seconds pass before you hear the bang. How many seconds sooner did a person below the waterline hear it? How far apart are the ships?

6. If the icecaps all melt, how many inches would the sea-level rise?

A Simple Task Scenario

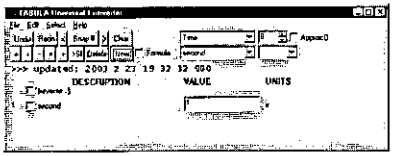
Let's consider the first task:

1. exercise to burn off a pound of fat. How many calories is that?

Let's make lines for "how long" and "100W light bulb". Time is a basic measure and quickly found as the third item in the top combo. We'll take any item in the combo below it (viz. second). You can always change it later, to minutes, hours, days... whatever TABULA knows about.

Click button: New...

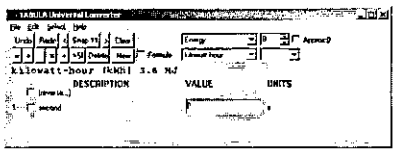
...line 1 appears.



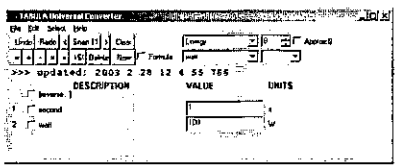
Now you can search in the two leftmost combos for a 'wattage' item, but there are faster ways to do the same thing. Search for "watt" in the calculator's data base. The Edit menu is the normal place to look for a 'Find' item, but now you've got a line with a VALUE field, you don't need it. You can also put single-letter commands in the VALUE column. These commands are followed immediately by their argument. Thus to search for "watt" you'd enter:

```
Fwatt
```

...the two item combos come up with the first match.



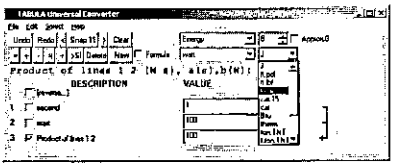
You don't happen to want this match. Press F3 (c/f Windows 2000 Notepad: it's the menu item: Edit / Find Next F3) to cycle round all the matches. When you find the one you want, click button: New... and enter: 100 (for the 100W light-bulb).



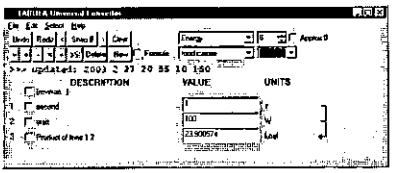
The product of these two lines should be "watt-seconds", an energy measure like "kilowatt-hours". To form the product, select both lines (by clicking the two checkboxes) and click the Product (×) button. The new line that appears has nominal units: W s.

What's a W s? It is indeed an energy unit, as TABULA will presently confirm. As soon as you click the checkbox of line 3 then the lower right combo gets a list of alternative compatible units that TABULA has met from its imported TXT files.

You see the first one (J) appear. Drop the combo down and select "kcal" (for food calories):



...line 3 is still the product of lines 1 and 2, but now expressed in the new energy units.



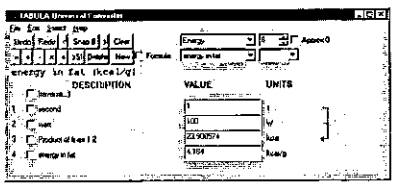
How much energy in a pound of fat? Let's look for the word "fat", using the 1-letter command: F or: f as we did before. In any VALUE field enter:

```
Ffat
```

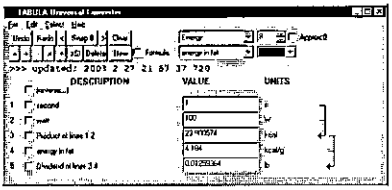
...the two line-selection combos locate a suitable constant.

Click button: New:

...the new line appears.



Now dividing line 3 by line 4 will give units of grams. Check lines 3 and 4, then click button: Divide (÷). Then check line 5 as you did with line 3 and change its units to: lb.



Now you might think to increase the value of "second" (line 1) in steps until line 5 approaches 1 lb. TABULA lets you do that, and indeed provides good support for it, letting you increase or decrease the entire number, or a selected digit position. Place the cursor in line 1 and press/hold ']' to increase the value typamatically, or '[' to decrease it ('[' ']' are used as arrow keys, since other alternatives are already in use). Watch line 5 approach: 1lb (it's fairly fast!).

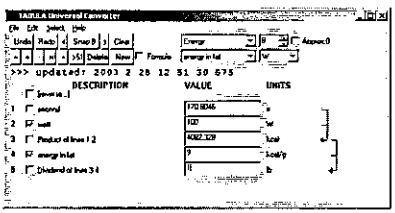
But TABULA has another trick up its sleeve. You can alter line 5 itself directly to the target value you want (viz. 1 for the 1lb you're asked for) and the alteration will be propagated back to the lines it depends on. We call this: *function inversion* because we have defined the forward function to calculate line 5 (L5) solely by means of two equations:

L5 = L3 / L4
L3 = L1 * L2

Yet TABULA is able, by a combination of the differential calculus and Newton-Raphson, to compute the inverse function: L5 → {L1, L2, L3, L4}. The algorithm used is powerful and general, and will be the subject of a separate paper, to be published some time in 2004.

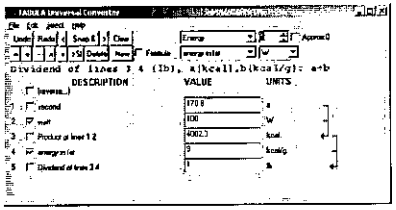
Notice that if you let TABULA do this, all four lines (1-4) will potentially change. Yet you want to keep the light-bulb wattage constant, not to mention the fat energy. To fix, or *hold*, the values of lines 2 and 4, letting only lines 1 and 3 change, check the boxes: 2 and 4, then enter a new number (viz. 1) in line 5:

...all fields are recomputed.



...and you can see the bulb will burn for 170.8046 s.

If you don't want to see all those figures after the decimal point, round them off in all fields by lowering the digit in the top right control from 8 to, say, 5 (meaning 5 significant figures overall, scientific-style, not 5 places of decimals):



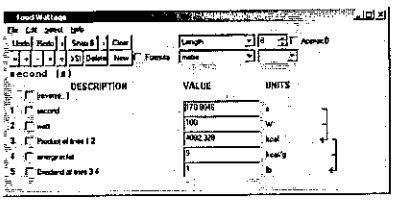
Now click button: Snap to save a snapshot of the form in memory. Each snapshot is a specialised calculator. All your current snapshots will be saved in a single named file when you choose menu: File / Save Ctrl+S in the normal way. You can entitle individual snapshots, their titles will appear in the Select menu, and as the window caption when you select one of them.

To entitle this snapshot "Food Wattage" you use another 1-letter command: T, typing it into any VALUE field. You'll notice the original numeric contents are instantly restored once the command has been executed. Thus, enter:

```
TFood Wattage
```

(the T is case-insensitive, but upper/lower case is honored in the argument.)

...whatever follows T becomes the window caption and the VALUE field regains its numeric value.



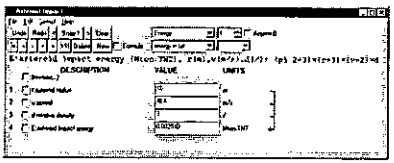
Finally replace the current snapshot (to update the window caption which is saved with it) by menu Select / Replace Snapshot (Alt+R), and save to disk (Ctrl+S).

From this simple scenario, you see that TABULA has a crop of useful features for creative calculation which saves you from having to type-in and parameterize formulas at run-time, or look-up units conversions. You can define new stored

constants and formulas in TXT files, which TABULA loads when it starts up. You don't have to specify formulas for the backward computations either, it works all that out for itself. It would be easy in principle to accept novel line definitions during a run session, but I haven't bothered to implement the input dialogs to accept all that (the hardest part) because it's just as easy to save your work, edit the TXT files and restart TABULA. Moreover this gives you greater flexibility when composing new formulas because you can consult existing formulas to help you.

Defining Constants, Conversions and Formulas

The syntax for defining new formulas is straightforward. Here's a fairly complex calculation, followed by the TXT file entries originally used to define each line (noting that the units and values have subsequently been redefined). You can display the formula for line 4 by placing the cursor in its VALUE field, as the screen-shot shows.



```
r:asteroid radius (m) 1 m
v:speed (m/s) 40000 m/s
d:relative density (/) 3 /
E:asteroid impact energy (Mton.TNT), r(m),v(m/s),d(/):
      (pi 2÷3)×(r*3)×(v*2)×d
```

You can write the TXT definitions either in APL syntax (I tell TextEdit to use the APLFONT screen font), or in an Excel-like syntax if you want to avoid APL symbols. For each formula TABULA makes a guess about which you've used (you'll see I've used APL symbols: ×*· above, augmented by a small repertoire of built-in arithmetic functions like pi and sqrt.) The above snapshot was created simply by selecting the formula: the preceding three input lines: r, v and d, were included automatically. You can specify existing lines as inputs to the new formula line by checking their boxes. TABULA matches them up by looking for compatible units, so you don't have to say which is which. If TABULA can't match your checked lines with formula variables, it automatically creates any extra input lines the formula needs.

The syntax for defining a constant or a conversion is:

```
[<var>:]<name>(<nominal-units>) <value> <units>
```

and for defining a formula is:

```
[<var>:]<name>(<nominal-units>),<dependency>:<formula>
```

<dependency> is:

```
<var>(<units>)[,<var>(<units>)...]
```

<nominal-units> serve as a new-unit definition provided by you in the TXT file,

<units> must be defined elsewhere in some other TXT line,

<value> can be an expression – any that APL can evaluate – but typically just a suggestive number representation, like that used for: grain, troy below, where 1÷480 expresses the fact that there are 480 grains in a troy ounce.

[<var>:] at the front of a definition is used for determining dependencies, but is not essential, except when specifying default input lines to accompany formula definitions in the TXT files.

Ultimately TABULA converts all units to SI-units, and it is marginally quicker to start-up if you define <units> in SI-units directly. However that is inconvenient and hardly ever done in my set of definitions. See my UMASS.TXT file:

```
gram                          (g) 1E¯3 kg
tonne                         (t) 1E3 kg

ounce, avdp                  (oz) 28.3495 g
ounce, troy              (oz.troy) 31.1035 g
grain, troy              (gr.troy) 1÷480 oz.troy
pound, avdp                  (lb) 16 oz
stone, avdp                  (st) 14 lb
hundredweight, avdp         (cwt) 112 lb
ton, avdp                   (ton) 2240 lb
ton, avdp                   (ton) 1.01605E3 kg
slug                       (slug) 14.5939 kg

deadweight ton              (dwt) 1 t
long tonne                (t.long) 1 ton
metric ton                 (ton.m) 1 t
bushel, wheat, USA    (bu.wheat) 60 lb
bushel, barley, USA  (bu.barley) 48 lb
bushel, oats, USA      (bu.oats) 32 lb
bushel, rye, USA       (bu.rye) 56 lb
```

```
atomic unit of mass       (a.u) 1.660430E¯27 kg
mass of neutron       (neutron) 1.674920E¯27 kg
mass of proton         (proton) 1.672614E¯27 kg
mass of electron     (electron) 9.109558E¯31 kg
```

Which is more or less as they appeared in the sources I looked-up [2], [4], hence the capricious use of g, kg, oz, lb, t and ton.

Incidentally you can use non-APL symbols in values, thus: `1/480, 9.109558E-31`. You can also define aliases. Thus: metric ton (ton.m) is an alias of: tonne (t), as is: deadweight ton (dwt).

You'll notice that: ton, avdp (ton) has been defined twice, once as 2240 lb and once as 1.01605E3 kg. Clearly the first definition is the clearer and happens to be exact, the second 'definition' is properly an approximation. It's only there for reference and experimentation. TABULA uses the first definition it comes across, because it looks up a list of the nominal units (of which ton is one) using {iota}. There's no requirement to define a unit in terms of units already defined – TABULA is multi-pass when it converts units.

No distinction is made between a constant and a units-conversion. One consequence of this is that you have the option of defining new 'units' (with a distinctive name) for each constant you specify. You needn't, but then you must make the nominal unit the same as the actual unit. Thus: mass of electron (electron) – from the above – will be recognised as a unit of mass in its own right. The same goes for earth and moon-masses, defined elsewhere. TABULA likes SI conventions, but doesn't enforce them.

Dimensional Analysis

TABULA internally reduces arbitrary units to basic SI units. There are only 7, viz:

```
Length, metre                (m) 1 m
Mass, kilogram              (kg) 1 kg
Time, second                 (s) 1 s
Current, ampere              (A) 1 A
Temperature, kelvin          (K) 1 K
Intensity, candela          (cd) 1 cd
Amount of matter, mole     (mol) 1 mol
```

It also converts a 'units' string to a canonical form, enabling it to recognise whether two given units are compatible, i.e. one can be converted into the other by a factor. Incidentally, under this definition, Celsius (°C), Fahrenheit, (°F) and

Kelvin (K) are *not* compatible units. They have to be linked in TABULA by formulas (which every schoolchild in England used to know by heart).

Notice the TABULA convention for a unitless measure: (/). The way units are converted, and different units expressions reconciled, the invention of (/) was rather like the invention of Zero in the Arabic number system. It is strictly a departure from SI. But so is the use of '/' anyway. The trouble is that the SI system uses constructs like this, which are complicated to handle in a computer program:

$$1\,J = 1m^2\,kg\,s^{-2}$$

In TABULA's canonical form, this becomes:

```
'J' matches 'm m kg/s/s'
```

where each basic SI unit is preceded by either a space or '/', serving both as separator and inversion operator. Either operator applies only to the unit immediately following it.

The purist might well want TABULA to reformat its canonical form to proper SI form when displaying units to the user – but it doesn't. Isn't this a purely cosmetic requirement, on a par with having a proper symbol: π? Why APL never admitted π into its character set is something I never quite understood – it makes the language seem less than serious to an old-fashioned physicist like me. No doubt there are APLers alive who remember the actual committee meetings at which this was thrashed out – and π lost! I suppose it would have had to be a postfix operator: the only one, because if you're going to admit π you really have to admit 2πr and 2π. Insisting on 2π1 or π2 just won't do. The TABULA built-in function: pi is more than just a cover-function to Circle: it allows (2 pi r) as well as (pi r) and (r pi 2) ...but not of course (r × 2 pi). It's amazing the amount of time you can waste over such trivia.

Calculating Results

The active lines of the display are generated from a single array, T1, which has a row for every numbered line. The contents of T1 defines the state of the system, i.e. screen updates are moved into T1, all computation is carried out in T1, and the results are moved back to the screen. The columns of T1 are never accessed in the code by in-line integers, always by named constants (APL doesn't give you named constants, so these are global vars!). These are:

Tname	the descriptive name of the line, can be altered by a 1-letter command: N
Tquan	the quantity appearing in the VALUE field
Tunit	the "nominal units" of Tquan as it appears in the UNITS column
Tsiqn	the quantity Tquan converted to SI units using: Tfact
Tsiun	the units of Tsiqn (always canonical SI units)
Tfact	the factor to convert Tquan to Tsiqn
Tdepn	a dependency vector, valid only for calculated lines, listing the input lines to the formula
Tfmla	the formula (source) for a calculated line: the actual formula used is compiled into an APL fn.
Tmodl	a "model" for function-inversion if the default model won't work. (Not explained here.)

Here's the contents of T1 for our Food Wattage example (the last column: Tmodl has been omitted):

```
      ct
   ⍳ Tname Tquan  Tunit  Tsiqn   Tsiun        Tfact  Tdepn  Tfmla
   1 secon 170.8  s      170.8   s            1
   2 watt  100    W      100     m m kg/s/s/s 1
   3 Produ 4082.3 kcal   17080   m m kg/s/s   4.184    1 2  a(s),b(W): a×b
   4 energ 9      kcal/g 37656   m m/s/s      4184
   5 Divid 1      lb     0.45359 kg           0.45359  3 4  a(kcal),b(kcal/g):
a÷b
```

All the end-user calculations (i.e. those defined by the set of formulas governing the calculated lines) are carried out in column Tsiqn. The steps of the calculation are:

- Fetch changed values from the screen into Tquan
- Convert Tquan to Tsiqn
- Compute new Tsiqn for each computed line, in a sequence governed by Tdepn
- Convert Tsiqn back to Tquan
- Update the screen to reflect Tquan.

The formula is not used directly to perform calculations. Instead each formula is compiled into a forward function which is executed to effect the forward calculation step. Here is the forward function for line 4 in the asteroid example, together with its source formula:

```
E:asteroid impact energy (Mton.TNT), r(m),v(m/s),d(/):
     (pi 2÷3)×(r*3)×(v*2)×d

    ∇ Δz←Δexe4;r;v;d
[1]   (r v d)←T1[ 1 2 3 ;Tsiqn]
[2]   Δz←(pi 2÷3)×(r*3)×(v*2)×d
    ∇
```

Drawing & Handling Dependencies

Column Tdepn of T1, the dependencies, govern not only the order of recalculation, both forward and backward, but the drawing of arrows on the screen to show what quantity feeds into what.

Tdepn is valid only for a calculated line. It is then a vector of line numbers, which map lines onto variables used in the formula according to the formula's dependency list.

Thus in the T1 listed above, Tdepn in line 5 is: (3 4). This says that var: a in the formula comes from line 3, and var: b comes from line 4. When new lines are added or deleted, Tdepn is easy to recompute.

In use, the Tdepn column is converted into a dependency matrix: dm, a boolean n-by-n matrix where n is the number of lines defined. As an example of how this is used, the execution sequence of forward functions is computed by:

```
z←sor ⍎closure dm
```

where fn: sor sorts columns of the (n,n) boolean matrix into dependency order. The fn: closure is computed by essentially OR-accumulating the matrix product of dm with itself until there is no further change:

```
z←z∨z∨.∧d
```

where z starts off as a copy of d.

Future Features

A proper Windows product is primarily menu-driven, an architectural imperative which Windows inherited (don't say "stole!") from prior WIMP (Windows / Icons / Mouse / Pointer) interfaces like the Xerox ALTO and the Apple Macintosh. This commands that all the functionality should be accessible from the menus alone. TABULA violates this commandment, notably with the 1-letter commands, none of which yet have corresponding menu items, nor the dialog boxes they'll need. This is cumbersome to program and also to use, being only useful to novices, which is not to say it isn't important. But the truth of the matter is that entering 1-letter commands into any handy VALUE field is the slickest way to drive TABULA.

Excel output is planned, and will be very useful. The existing crop of snapshots in memory will be saved as a single workbook having one snapshot per sheet. Thanks to MSO-view, this enables TABULA to be used to generate more specialised calculators like those cataloged in [3].

Graph plotting is an obvious enhancement, as is a 2-D table. TABULA will allow the column of figures it builds to be replicated horizontally as a 2-array, stepping one or more of the lines. This leads naturally to graphic output. An existing chart package will be bolted-on, or else the 2-array will be output to Excel, allowing chart sheets to be defined by the Excel user.

References

[1] The Sinclair Executive pocket calculator:
http://www.nvg.org/sinclair/calculators/executive.htm

[2] Seafriends - Units, constants and conversions :
http://www.seafriends.org.nz/books/units.htm

[3] Martindale's The Reference Desk – Calculators On-Line Center: http://www-sci.lib.uci.edu/HSG/RefCalculators3A.html

[4] Kaye & Laby, Tables of Physical and Chemical Constants, 14th Edition (1973), Longmans.

Googling with APL+Win

by William R. Parke (wrp@parkenet.org)

Introduction

The Google Web API allows developers to incorporate customized access to Google's Internet database in their applications. It also provides a simple and useful example for learning how to use Extensible Markup Language (XML). The purpose of this article is to describe the essential steps in accessing the Google API from APL+Win, including techniques for handling XML documents and transmitting them over the Internet, and some comments on developing Google API search applications.

In presenting this information, it is assumed that the reader has some familiarity with XML notation and the APL+Win `⎕wi` system function, including creating ActiveX objects, using object notation, and setting and referencing object properties.

The Google Web API

Google publishes an Application Programming Interface (API) that allows access to their Internet database with three search methods:

doGoogleSearch

the standard Google query operation

doGetCachedPage

returns Google's cached version of a Web page

doSpellingSuggestion

returns a suggested spelling for words and names in Google's database

(this is what you see when Google says "Did you mean: ..." at the top of the page when you mis-spell a search term)

Each method's request and response are sent to and returned from Google.com as SOAP messages which are specially-conforming XML documents. The XML elements in these messages identify the parameters of the search request and the named data values comprising the search results. Since these are simply text strings, they can be constructed or parsed easily in APL.

Four processes are needed to implement the Google API:

1. **Obtain a Google user-authorization key and developer toolkit**

   The first step in using the Google API is to download the developer's toolkit and obtain a free user-authorization key from Google. The key must be sent with each search request and entitles the user to 1,000 requests per day. Go to Google.com and follow the links to Services & Tools | Google Web APIs (as of this writing still www.google.com/apis/).

   It is helpful to look at the `APIs_Reference.html` document and the `\soap-samples\*.xml` request and response examples in the Google Web API toolkit. XML is essentially a method of labelling data, enclosing values in pairs of name-tags of the form *<name>value</name>*, and all the element names and hierarchies needed for the Google API are presented in these examples.

   Opening the soap-sample XML files in a Web browser is a good way to view them, since the browser will display the tree hierarchy that structures the XML document. For the SOAP request messages, the outer elements, `<SOAP-ENV:Envelope>` and `<SOAP-ENV:Body>` are the same in all methods. The elements of consequence are contained in `<ns1:doGoogleSearch>`, `<ns1:doGetCachedPage>` and `<ns1:doSpellingSuggestion>`. In the response messages, the `<return>` element contains Google's results for each method.

2. **Implement methods for handling XML**

   XML can be manipulated easily in APL, either with an XML parser object or with traditional APL character-vector methods. For simplicity, all the methods described in this article use Microsoft's MSXML DOMDocument object which should be available on any Windows-based system with Internet Explorer 4 or above installed. MSXML3 currently is the version installed with Internet Explorer. You may prefer working with MSXML4 which has some improvements and can be obtained online from Microsoft. If you decide to install it, I recommend getting the version with the optional SDK which you'll find helpful once you get hooked on XML. However, MSXML version 3 has all the functionality you need for this project.

3. **Connect your application to the Internet**

   To connect to Google.com from your application, you can use an MSXML HTTP object, as shown in the examples below, or APL+Win's `Dni` sockets. AplWebServices or other 3rd-party socket managers also can be used to handle the connections.

4. **Manage the search results**

   What do you want to do with collections of search results from Google? There are many possible uses for API-managed searches, but the principle benefit is the ability to archive and work with the data over time. Some comments on creating search applications will be offered below.

Gentlemen, Start Your (Search) Engines

It is convenient to prepare XML templates for each Google API search method, either as variables in the workspace or as external files, by taking the examples that come with the developer toolkit and removing the content (values) from the search-parameter elements. The templates then can be loaded into the MSXML object and "filled-in" with the values specific to each search request. To illustrate, let's start with the simplest API method, *doSpellingSuggestion*.

Open the file *doSpellingSuggestion.xml* from the API toolkit and read it into a character vector in APL (or copy and paste from a text editor), then remove everything between the *<key xsi:type="xsd:string">* and *</key>* tags and the *<phrase xsi:type="xsd:string">* and *</phrase>* tags. Save the changes to a reusable (global) variable or to a file.

Next, create an MSXML DOMDocument object:

```
'xmlobj' ⎕wi 'Create' 'MSXML2.DOMDocument'
```

This will default to version 3 even if version 4 is installed. If you have MSXML4 installed and want to use it, create *'MSXML2.DOMDocument.4.0'* instead to force version 4.

Now load the XML document into the MSXML object:

```
⍝ string variable:
'xmlobj' ⎕wi 'loadXML' xml_doSpellingSuggestion
⍝ or file name:
'xmlobj' ⎕wi 'load' 'c:\googleapi\soap-samples\doSpellingSuggestion.xml'
```

On success, the result of the method-call is 1, otherwise the document must contain an XML validation error. As an aside, you may find it useful to place the following statements in a function or user-command to help track down these errors, especially if you are generating XML from code:

```
'Error: ', 'xmlobj' ⎕wi 'parseError.reason'
'Line : ', 'xmlobj' ⎕wi 'parseError.line'
'Char : ', 'xmlobj' ⎕wi 'parseError.linepos'
```

MSXML object methods are used to set the content of the XML document elements specific to a particular search request. In MSXML syntax, an XML element is referenced as a "node" in the tree hierarchy of the parsed XML document, and the MSXML method *selectSingleNode* is used to specify an XML element or "node" to set or reference. The argument to *selectSingleNode* is an XPath expression, which in its simplest form is an XML element name, most

often prefixed with '//'to reference it relative to the document root. (There are many other ways to identify a specific node. You can learn more about XPath in the MSXML SDK or other resources. Just be aware that XML is case-sensitive, so you must specify element names exactly as they appear in the examples.) The result of the *selectSingleNode* method is an object representing that element's node in the document tree, and the node object's *text* property contains the element's value or content.

In the case of the *doSpellingSuggestion* SOAP message, only two parameters are required: the Google user-authorization key, and the word, name or phrase to check for spelling. For the example below, I have created a string variable *google_user_key* that contains my user authorization key. I use this in *⎕wi* object notation to set the *text* property of the document node object representing the *key* element. By doing this, I am setting the content of the XML *<key>* element in the document.

```
      'xmlobj' ⎕wi 'selectSingleNode("//key").text' google_user_key
      'xmlobj' ⎕wi 'selectSingleNode("//phrase").text' 'apl sftwar'
```

Now if you reference the node's text property, the value is shown to be what was just set:

```
      'xmlobj' ⎕wi 'selectSingleNode("//key").text'
EGJk8grQ1jL5LHqs

      'xmlobj' ⎕wi 'selectSingleNode("//phrase").text'
apl sftwar
```

(In general use, you would want to wrap such methods in APL functions which validate the existence of the specified node object before attempting to set or reference it.)

You can can inspect the completed XML document string by referencing the XML object's *xml* property:

```
      document ← 'xmlobj' ⎕wi 'xxml'
```

Note that this is a read-only property. If you manually edit the string variable outside the MSXML DOM object, you must use the *loadXML* method to place it back into the object, as shown above.

Once the SOAP request is set, it can be sent to Google, in this example by using an MSXML ServerHTTP object (if you have MSXML4 installed, you can create *'MSXML2.ServerXMLHTTP.4.0'* instead).

```
'httpobj' ⎕wi 'Create' 'MSXML2.ServerXMLHTTP'
⍝                    method  url                                          async
'httpobj' ⎕wi 'open' 'POST' 'http://api.google.com/search/beta2' 0
⍝ send the property of the XML DOM which contains the SOAP request string
'httpobj' ⎕wi 'send' ('xmlobj' ⎕wi 'xxml')
```

The third argument to *open*, 0 (zero), turns off asynchronous mode, causing the XMLHTTP object's *send* method to wait until it connects and receives a response. When the *send* method returns, you can check its *status* property, then retrieve Google's SOAP response and handle it with the XML DOM object or APL code. For *doSpellingSuggestion*, the response is a simple string, or nothing if Google has no suggestion:

```
:if 200 = 'httpobj' ⎕wi 'status'
    ⍝ transfer the returned SOAP message to the XML DOM object
    :if 1 = 'xmlobj' ⎕wi 'loadXML' ('httpobj' ⎕wi 'responseXML.xxml')
        ⍝ extract the response from the <return> element
        spelling ← 'xmlobj' ⎕wi 'selectSingleNode("//return").text'
    :endif
:endif
```

That's all there is to the process. The *doGetCachedPage* request differs only in that it sends a URL instead of a phrase, and Google returns a base-64 encoded string which must be converted to HTML text. Following the example above:

```
'xmlobj' ⎕wi 'loadXML' xml_doGetCachedPage
'xmlobj' ⎕wi 'selectSingleNode("//key").text' google_user_key
'xmlobj' ⎕wi 'selectSingleNode("//url").text' 'www.apl2000.com/new.htm'
 ... XMLHTTP steps not shown as they are exactly the same as above ...
html ← fromBase64 'xmlobj' ⎕wi 'selectSingleNode("//return").text'
```

doGoogleSearch

doGoogleSearch is more complicated. It takes ten request parameters and returns ten general data elements describing the search results plus ten more elements for each matching result item. To use this method, load the XML object with a *doGoogleSearch* template and set the search parameters:

```
    'xmlobj' ⎕wi 'loadXML' xml_doGoogleSearch
    'xmlobj' ⎕wi 'selectSingleNode("//key").text'          google_user_key
    'xmlobj' ⎕wi 'selectSingleNode("//q").text'            'apl'
    'xmlobj' ⎕wi 'selectSingleNode("//start").text'        '0'
    'xmlobj' ⎕wi 'selectSingleNode("//maxResults").text'  '10'
    'xmlobj' ⎕wi 'selectSingleNode("//filter").text'       'true'
    'xmlobj' ⎕wi 'selectSingleNode("//safeSearch").text'  'true'
    'xmlobj' ⎕wi 'selectSingleNode("//restrict").text'    'linux.(countryFR|coun
tryBE)'
    'xmlobj' ⎕wi 'selectSingleNode("//lr").text'          'lang_fr|lang_de'
    (<ie> and <oe> are required but ignored by Google, so just send defaults)
```

Note that the search elements may be set in any order, and numeric values must be specified in character format. The query string element *<q>* may include various qualifiers such as double-quotes, exclude (-), and several filtering terms such as *intitle:*, *site:*, *daterange:*, etc. In addition, Google accepts country, language and topic restrictions to further refine the results. The *APIs_Reference.html* document in the API toolkit explains all these terms and how to use them to modify searches.

After sending the request to Google, the response data are retrieved from the child elements contained in the *<return>* element. You can reference them in any order, or skip ones you're not interested in. Here is the complete list:

```
q ← 'xmlobj' ⎕wi 'selectSingleNode("//searchQuery").text'
r ← 'xmlobj' ⎕wi 'selectSingleNode("//estimatedTotalResultsCount").text'
x ← 'xmlobj' ⎕wi 'selectSingleNode("//estimateIsExact").text'
t ← 'xmlobj' ⎕wi 'selectSingleNode("//searchTime").text'
s ← 'xmlobj' ⎕wi 'selectSingleNode("//startIndex").text'
e ← 'xmlobj' ⎕wi 'selectSingleNode("//endIndex").text'
f ← 'xmlobj' ⎕wi 'selectSingleNode("//documentFiltering").text'
p ← 'xmlobj' ⎕wi 'selectSingleNode("//searchTips").text'
c ← 'xmlobj' ⎕wi 'selectSingleNode("//searchComments").text'
```

One additional element, *<directoryCategories>*, must be handled differently, since it may contain zero or more *DirectoryCategory* arrays as child elements. You can determine how many arrays are present by checking the *arrayType* attribute which looks like this: *ns2:arrayType="ns1:DirectoryCategory[2]"*

```
n ← 'xmlobj' ⎕wi 'selectSingleNode("//directoryCategories").
      getAttribute("ns2:arrayType")'
n ← ((-⎕io)+n⍳'[')↓n
n ← ⎕fi ((-⎕io)+n⍳']')↑n
```

However, it is easier to simply loop on an MSXML *selectNodes* object which references all matching XML elements as a group. In this instance, it is useful to create additional temporary named objects for use with *⎕wi*:

```
:if 0 ≠ 'xmlobj' ⎕wi 'selectNodes("directoryCategories/item")'
    'xmlobj' ⎕wi 'selectNodes("directoryCategories/item") > snodes'
    :for i :in (-⎕io)+⍳'snodes' ⎕wi 'length'
        'snodes' ⎕wi 'Item(',(⍕i),') > inode'
        encoding ← 'inode' ⎕wi 'selectSingleNode("specialEncoding").text'
        viewname ← 'inode' ⎕wi 'selectSingleNode("fullViewableName").text'
        ⍝ ... do something here with each data pair ...
    :endfor
:endif
```

doGoogleSearchResponse may contain up to ten *<resultElement>* items representing the "hits" or Web pages matching the search query. To extract their contents, run a loop using the MSXML *selectNodes* method and named objects as described above.

```
:if 0 ≠ 'xmlobj' ⎕wi 'selectNodes("//return/resultElements/item")'
    'xmlobj' ⎕wi 'selectNodes("//return/resultElements/item") > snodes'
    :for i :in (-⎕io)+⍳'snodes' ⎕wi 'length'
        'snodes' ⎕wi 'Item(',(⍕i),') > inode'
        title    ← 'inode' ⎕wi 'selectSingleNode("title").text'
        url      ← 'inode' ⎕wi 'selectSingleNode("URL").text'
        size     ← 'inode' ⎕wi 'selectSingleNode("cachedSize").text'
        host     ← 'inode' ⎕wi 'selectSingleNode("hostName").text'
        snippet  ← 'inode' ⎕wi 'selectSingleNode("snippet").text'
        rip      ← 'inode' ⎕wi
            'selectSingleNode("relatedInformationPresent").text'
        dirtitle ← 'inode' ⎕wi 'selectSingleNode("directoryTitle").text'
        summary  ← 'inode' ⎕wi 'selectSingleNode("summary").text'
        encoding ← 'inode' ⎕wi
            'selectSingleNode("directoryCategory/specialEncoding").text'
        viewname ← 'inode' ⎕wi
            'selectSingleNode("directoryCategory/fullViewableName").text'
        ⍝ ... do something here with each result set ...
    :endfor
:endif
```

Another way to do this is to reference a *selectNodes* object for each data item of interest:

```
urls ← ⍬
:if 0 ≠ 'xmlobj' ⎕wi 'selectNodes("//URL")'
    'xmlobj' ⎕wi 'selectNodes("//URL") > snodes'
    :for i :in (-⎕io)+⍳'snodes' ⎕wi 'length'
        urls ← urls, ⊂ 'snodes' ⎕wi 'Item(',(⍕i),').text'
    :endfor
:endif

sizes ← ⍬
:if 0 ≠ 'xmlobj' ⎕wi 'selectNodes("//cachedSize")'
     'xmlobj' ⎕wi 'selectNodes("//cachedSize") > snodes'
     ⍝ as above ...
...
```

There are other ways to work with MSXML objects, depending on the needs of your application, but for the Google API, these methods should be sufficient.

In Search Of...

Missing from these examples is what to do with the search data obtained. The results could be displayed as an HTML page similar to Google's own browser format by using an XML Stylesheet Transform (XSLT) or generating HTML directly from APL code. Of more value would be to organize and save the data so that topics of interest could be researched offline and tracked for changes over time. You might like to study changes in page rankings or compare newer and older versions of selected Web pages. Any search query you type in more than once would be a good candidate for an API-managed search and archival system, even more convenient if special search terms or filters are used. But no matter how sophisticated your search techniques, still a fair number of the results Google returns often are not applicable to your needs. The ability to build, save and update custom result databases could be quite useful and timesaving if you didn't have to sift through pages of unwanted items each time you revisited a topic.

Beyond the Google API, the techniques for working with XML documents are interesting and useful by themselves. Once you start, you see uses for XML everywhere. It is ideal for configuration files, hierarchical data, and any situation where portability of named values is desired. As XML becomes the standard for data exchange, it should be well worth the time and effort to learn the rudiments of this format. Try it out with the Google API.

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

At Play with J: Jacob's Ladder

by Eugene McDonnell

And he dreamed, and behold a ladder set up on the earth, and the top of it reached to heaven: and behold the angels of God ascending and descending on it.
Genesis 28:12

Dedicated to my grandson Jacob (15 months old)

The Name of the Game

Lewis Carroll invented a game he called *Doublets* in 1879. He used *doublet* to describe two words of the same length, which were to be connected by a *chain* of other words, called *links*. Two words are linked if they use the same letters in every position but one, like *rota* and *iota*. As an example, he gave as a doublet the words *head* and *tail*, and for the links the words *heal, teal, tell* and *tall*, so that the entire chain was *head, heal, teal, tell, tall* and *tail*. The word *Doublet* hasn't stuck, however, and the game is now usually called *Word Ladders*.

The Word Ladder game can be played mentally, and many people enjoy playing in their head, sometimes making it a game for two or more people, to see who can find a ladder quickest. This article, however, treats computer solutions to the problem, which now asks that the chain be as short as possible. There may be more than one shortest solution. For example,

```
+----+----+
|head|head|
|heal|heal|
|teal|heil|
|taal|hail|
|tail|tail|
+----+----+
```

are two solutions shorter by one link than Carroll's. Carroll would probably point out that *taal* is usually capitalized, in phrases like *the Taal*; it is a name for a language, like *English* or *Italian*, and is another name for *Afrikaans*, one of the official languages of South Africa; and *heil* is a German interjection used infamously by the Nazis in phrases like *Heil Hitler*. These words let me point out that *all* the words in my word tables are in lower case, even names like *Hugo* and *Clive*; and that they include numerous words from foreign languages that have gained currency in English, like Russian *dvor* and Spanish *amigo*. Different word lists will give different results.

Some doublets give rise to eight or more solutions – here are the solutions for the doublet *white* and *black*.

white	white	white	white	white	white	white	white	white
whine	whine	whine	whine	whine	whine	whine	whine	whine
chine	chine	chine	chine	chine	chine	chine	chine	chine
chink	chink	chink	chink	chink	cline	cline	cline	cline
chick	clink	clink	clink	clink	clink	clink	clink	clink
click	blink	clank	clank	click	blink	clank	clank	click
clack	blank	blank	clack	clack	blank	blank	clack	clack
black	black	black	black	black	black	black	black	black

Later on I'll bring to your attention the ladders for the doublet *dvor* and *lade*: there are *eighty* solutions to it, each nine words long!

My word tables have a history. Many years ago, in our IPSA office in Palo Alto, Joey Tuttle acquired a tape from Houghton Mifflin, the publishers of *The American Heritage Dictionary*, that contained an alphabetical listing of all the words in their dictionary. Joey extracted a number of files, each file giving all of the words having the same length, and mounted these on the I. P. Sharp computer in Toronto. I found the list useful in a number of ways, and one of the things I did was to write an APL program to form Word Ladders, of which more later. I have these word files now on my personal computer.

Structure of the Game

Looked at from the point of view of the game, a table of words is seen as an undirected graph, where the nodes are the words, and the edges are the links for each word. Linkness is symmetric: if *iota* is a link of *rota*, then vice-versa. This being the case, the graph is also symmetric, and the counterdiagonal is all zero – a word is not a link of itself. A word may have many links. For example, *bare* has 26, using my table. They are:

babe	bake	bale	bane	base	bate	barb	bard	bari	bark	barm	barn	bars
care	dare	fare	hare	mare	pare	rare	tare	vare	ware	yare	bore	byre

A word may have no links, as well. Some four-letter hermits are *agog*, *ecru*, *idol*, *ugly*, *xmas*, *yeti* and *zarf*.

I've adopted some naming conventions. A word list - actually a character table - is called *Tn*, where *n* is the length of the words in the table. Thus my four-letter word table is *T4*. A link list, one that gives the links for each word, is called *En*. Thus, the link list for four-letter words is *E4*. Later on, in the discussion of J functions, I'll introduce some more conventions.

The Link List

There's a story to do with the creation of link lists from a word table. I wrote one for myself that took a *Tn* as argument, and produced the corresponding *En*. I showed this to Roger Hui, who noted that it had quadratic time. He thought it would be possible to make one that had linear time. At the time of this message we were still calling link lists *neighbour lists*.

```
From:          rhui000@shaw.ca
Subject:       Re: Word Ladders
Date:          February 14, 2004 8:10:30 AM PST
To:            eemcd@mac.com
Reply-To:      RHui@Jsoftware.Com

There is indeed a much faster, linear, method for generating
the neighbors list.  The idea is this: for each word, blank
out successive letters, and use these blanked out words
to match for neighbors.  e.g. if the words are abba and abbe,

   _bba  _bbe
   a_ba  a_be
   ab_a  ab_e
   abb_  abb_

So if you have a (m,n) matrix of words, in the matching
you'd be dealing with a ((n*m),n) matrix and doing linear
operations on it, rather than the (m,m,n) outer product.

mlx =: <@I.@(<:@#@[ = +/@|:@:="1)"1 2~    NB. eemcd

mlx1=: [: <@I."1 <:@{:@$ = +/ @: ="1 / ~  NB. eemcd

mlx2=: 3 : 0                        NB. hui
 'm n'=. $y.                        NB. # of words, # letters
 i=. n (* + i.@[ *"1 -.@])=i.n      NB. indices for blanking successive
positions
 t=. ,/ i{"_ 1 y.,.'_'              NB. blanked words
 p=. ,/@:(([ ,. -.~)"0 1)~          NB. verb for pairing
 j=. ; t <@p/. n#i.#y.              NB. group word indices per blanked words
 h=. ({."1 </. {:"1) j              NB. unordered neighbors
 k=. ~. {."1 j                      NB. word indices corresp. to h
 (m{.h-.&.>k) /: k,(i.m)-.k         NB. reorder
```

```
)

mlx and mlx1 give identical results.  mlx2 gives neighbor
indices that are unordered, but is otherwise identical.
The improved efficiency in mlx2 is more pronounced as the
number of words gets large.

   alp=: a.{~97+i.26
   x=: ~. alp {~ 4000 4 ?@$ #alp          NB. table of 4000 'words'
   $x                                     NB. 3980 after duplicates removed
3980 4

   (mlx  -: /:~&.>@mlx2) x
1
   (mlx1 -: /:~&.>@mlx2) x
1

   ts=: 6!:2 , 7!:2@]                     NB. to find time and space used
   ts 'mlx x'
81.9158 608640
   ts 'mlx1 x'
12.1867 8.3887e7
   ts 'mlx2 x'
0.490401 1.47846e6
```

As you can see, for a table of 3980 words, Hui's program is about 160 times faster. That's a lot. It does use twice the space, but it's worth it. I've been happily using `mlx2` since to form my link lists. It's always a joy to get instant results rather than "start the process, go out and mow the lawn, then come back and maybe it will be done."

Two Schools of Thought

I'll discuss two different ways of building ladders, given a table `Tn`, and a links list `En`. The first way is the standard approach to finding paths in a graph: search forward from one of the words, the *starting* word, and work one's way through until the other word, the *goal* word is reached, or it is found that there is no path. This is the approach used in Edsgar Dijkstra's well-known algorithm. The function that uses this forward search I call `FL`, for "forward ladders".

The second way, one I used a quarter-century ago, started searching from both ends of the problem. This can be done because of the problem's symmetry: if I find the shortest path from `white` to `black`, I've also found the shortest path from `black` to `white`. My intuition told me that a forward search algorithm would take significantly more space, and possibly more time, than a two-way search. Somewhat fortuitously, the two-way search can be identified with the biblical

ladder in Jacob's dream, with angels going up and down. Because of this, and in honour of my grandson Jacob, I'll call the two-way search the *Jacob's Ladder* way, and the associated function `JL`, for "Jacob's Ladder ".

My reasoning was this: suppose we use `FL`, starting with a one-by-one matrix, and that there are three links in every item of `En`. After one step, we have at most a three-by-two matrix; after two a nine-by-three ... and after eight we have at most a 6561-by-9 matrix. Suppose that we have now found the goal. We've looked at 6561 nodes.

If, on the other hand, we use `JL`, after four iterations forward and backward, we have at most two 81-by-5 matrices, meaning that we have, at most, 162 nodes. In each case we've taken eight steps. The backward steps begin with a node that is necessarily one of the 6561. The next backward step continue with 3 nodes, one of which is in that of the 2187 of the forward search. Likewise, the next backward step finds 9 nodes, one of which is in the 729 of the forward search. Next 27, one included in the 243. By the next step, finding 81 nodes, the forward search has also found 81, and one of the backward 81 nodes must be among the 81 in the forward search. Thus the forward search has worked with 40 times more nodes than the forward and backward search. I had to conclude that it would take less time and less space for the two-way search.

The Case of the Mysterious Test

I wrote `FL` and `JL`, tested them, and found that my reasoning was correct, and communicated the results to Roger Hui. His response was dumbfounding. Earlier I had sent Roger a copy of the first fifty words in T4. He generated a two-column table called `pairs` giving all 1225 combinations of two out of fifty, and used this as the right argument in a timing test. He made the left argument `E` using `m1x2` on the first fifty words. This is what he saw:

```
   $pairs=: 2 comb 50
1225 2
   ts 'E FL"1 pairs'
0.94485 1.1991e6
   ts 'E JL"1 pairs'
0.90665 1.1991e6
```

I tried this test on my machine and got the same kind of result: the forward search had essentially the same speed and the same use of space as Jacob's Ladder. This was completely contrary to all the earlier measurements I had made, but I couldn't deny what my own senses told me. For several days I was at a standstill – I had no idea what was wrong with my earlier measurements – or, less

likely, whether there was anything wrong with Roger's measurements that I didn't understand.

The Case Solved

There was to be a happy ending however. Roger had earlier told me that he knew a way to do pathfinding that was yards better than mine, and not only that, but would find the shortest paths for all possible pairs. I asked him if he had experimented with this yet. Then this message came:

```
A few minutes of experimentation with the 4-letter words
reviews the fallacy of my approach.  The transitive
closure of that neighbors list took so long that I
interrupted it after about 10 minutes.  Further investigation
reveals that most words are reachable from most other words.
Starting from every way possible generates too much
information (and too much redundant information).

If I had to choose, I'd choose the your original FL
approach.  JL is not sufficiently faster enough to
warrant the extra complications.
```

This was interesting, but still left me baffled by the anomalous timing we both had seen. But then, the same day, came a new message:

```
Further … The up-and-down approach is enough faster after all…

   ts 'i=: E FL 704 1407'
11.9041 3.04531e6
   ts 'i=: E FL2 704 1407'
2.27514 1.66336e6
   ts 'i=: E JL 704 1407'
0.881336 515264
```

The test he now used, timing for the pair 704 1407, was one that had eighty(!) nine-node solutions. The first test above used my original `FL`; the second used `FL2`, his rewriting of `FL`; the third used my original `JL`. This was very welcome news, but what about those anomalous timings? How explain them? I looked at all aspects of the data and I believe I can now explain it.

The anomalous results occurred because the 1st 50 words had no link list longer than 5, and most were 3 or less; the average number of links per node was 1.8; 18 out of the 50 link lists were empty, a very large percentage; thus the timings reflected an atypical set, one in which the up-and-down program couldn't show itself better than the forward search.

The best demonstration is to compare the results of a test using E which has only 50 items, with tests using E4, which has 2962 items. Here are the tests using E, showing FL and JL essentially equal in time and space:

```
   ts 'E FL"1 pairs'
0.94485 1.1991e6
   ts 'E JL"1 pairs'
0.90665 1.1991e6
```

The only change in the next tests is using the E4 of 2962 nodes. This averages 8.1 links per node, as many as 31; 91 are empty, only 3.2%. The distribution of links in the first 50 items of E4 is also quite different from that of E. There are 155 links, an average of 3.1 per item, and one has as many as thirteen. Only seven are empty. Here are the tests using E4:

```
   ts 'E4 FL"1 pairs'
674.971 6.14182e6
   ts 'E4 JL"1 pairs'
24.1225 2.28742e6
```

To me, this is conclusive. With this one change, JL is 28 times faster than FL, instead of being equal. It uses less space, less than half as much as FL.

```
   675%24        NB. time ratio
28.125
   6.142%2.287   NB. space ratio
2.69
```

Just for the purpose of this paper I've made some more tests, using both E4 and E5, with right argument 100 random pairs of nodes, with no node repeated. The results are what I have come to expect to get.

```
Tests of 100 random pairs from T4 and T5, no node repeated,
2004 03 05
   $E4
2962
   pr4=: 100 2$200?#E4          NB. 200 distinct numbers from i.
2962
   f4 =: ts 'E4 FL"1 pr4'
   j4 =: ts 'E4 JL"1 pr4'
   f4,:j4
58.3915 4.19264e6               NB. FL numbers
4.08996 2.68454e6               NB. JL numbers
   f4%j4                        NB. ratio of test numbers
14.2768 1.56177                 NB. JL 14 times faster, 1.5
times less space
```

```
   $E5
5604
   pr5=: 100 2$200?#E5          NB. 200 distinct numbers from i.
5604
   f5 =: ts 'E5 FL"1 pr5'
   j5 =: ts 'E5 JL"1 pr5'
   f5,:j5
69.0011 2.84186e6               NB. FL numbers
4.43224    843456               NB. JL numbers
   f5%j5                        NB. ratio of test numbers
15.568 3.3693                   NB. JL 16 times faster. 3.4
times less space
```

Function Syntax and Use

The versions of FL and JL I wrote were gone over and tightened up by Roger Hui. Their syntax is:

```
R =: En FL a,b
R =: En JL a,b
```

Where a and b are indices of words in some Tn. The result is an integer table, where each row is distinct, and the successive values in a row are pairwise links, with first item a and last item b.

```
   ] R =: E4 JL 2182 861
2182  628  617  616 589 810 859 861
2182 1505 1483 1455 812 810 859 861
2182 2619 2608 2573 812 810 859 861
```

To obtain the desired word ladder, use this result as an index to T4:

```
<"2 T4 {~ R
```

For example,

```
   <"2 T4 {~ E4 JL 2182 861
+----+----+----+
|rips|rips|rips|
|dips|lips|tips|
|dies|lies|ties|
|died|lees|tees|
|deed|fees|fees|
|feed|feed|feed|
|fled|fled|fled|
|flew|flew|flew|
+----+----+----+
```

Forward March

The program FL is easier to explain than the more intricate JL.

```
FL =: 4 : 0
  's e'=. y.
  u=.d=.  ,s
  c=. ,.s
  while. -. e e. d do.
   d=. ; v=. (d{x.) -. &.> < u
   if. '' -: d do. _1 return. end.
   u=. u , ~. d
   c=. ((#&>v)#c) ,. d
  end.
  c #~ e=d
)
```

The two word indices are s and e. In FL, these signify start and end, but in JL they don't have that mnemonic significance, since they start separate chains. The variable x. is the links list, some En.

Variable d is an integer list, initially ,s. It is used to select boxed lists of potential new links. The first time through it gets all the links of s. Next time it gets all the links of those links, and so forth. Variable u contains all of the links already seen. No link appears in u more than once. Initially it is ,s. It is used to ensure that no later use is made of a link that has already been used. This is because once a link is used in any step, there is no point to using it again in a new step - any chain with a later appearance of an earlier link must be longer than one with an earlier appearance.

Variable c is the table of chains, initially with s as its only value. Within the while. loop it will be extended. Each of its rows represents a potential shortest path.

The while. loop continues until d contains e as an item; when it does, it means that one or more shortest paths have been found.

Variable d is used to select boxed lists of links from x.. Before further use, each box has removed from it all links that have already been used. These cleaned-up boxes are assigned to v, the raze of which becomes the new link selector list d.

If d is empty, there are no new links, and since e hasn't been found, we'll have to admit that there are no paths between s and e. When this happens, the scalar _1 is returned.

Variable u is updated with all the new links, by appending d to it. Because u should never contain two appearance of the same link, duplicates are removed from d before appending it.

Table c is updated by adding a new column, with the items of d.

When the `while.` loop ends, a selecting mask is formed by the equality of scalar e and list d, and this mask is used to remove from c all those rows not ending in e. This gives the desired result.

The Angels of God Ascending and Descending

Since JL goes forwards and backwards, there are separate variables for the forward and backward sequences. Instead of c we have sc and ec – two chains; instead of u and d we have su and eu, sd and ed.

The `while.` is different – it says, effectively, "while forever", since the loop continues as long as the value of the `while.` phrase is 1. The exiting from the loop will take place by way of `if.` statements.

```
JL =: 4 : 0
 sc=. ,. su=.sd=. ,{.y.
 ec=. ,. eu=.ed=. ,{:y.
 while. 1 do.
  if. +./ sd e. ed do. break. end.
  if. '' -: sd=. ; v=. (sd{x.) -.&.> <su do. _1 return. end.
  su=. ~.su,sd [ sc=. sd ,.~ (#&>v)#sc
  if. +./ sd e. ed do. break. end.
  if. '' -: ed=. ; v=. (ed{x.) -.&.> <eu do. _1 return. end.
  eu=. ~.eu,ed [ ec=. ed ,.  (#&>v)#ec
 end.
 sc join ec
)
```

The variables ending in u, d and c have the same functions as the u, d and c variables in FL. The first and the last three statements in the `while.` loop have almost identical structure. With two path tables being built in the same loop, the test for termination is by finding that the same item or items appear in sd and ed. The test for "no path found" is essentially the same as in FL, but there are separate ones for sd and ed. An important difference is that sc is built from left to right, but ec is built from right to left. This makes joining the two easier.

Linking the Chains

When the `while.` of JL terminates, the fitting together of the two path tables requires some agility. It is complicated enough so that a special `join` function has been made. It was written by Roger Hui as a rewrite of a `joinends` function provided to me by R. E. Boss when I sent a message to the J forum explaining the problem and asking for a solution.

```
join=: 4 : 0
 x.=. x.#~ ({:"1 x.) e. {."1 y.
 y.=. y.#~ ({."1 y.) e. {:"1 x.
 (({."1 i){}:"1 x.) ,. ({:"1 i){y.
     [ i=. (0,#y.)#:I.,({:"1 x.)=/{."1 y.
)
```

The variables `x.` and `y.` are the forward and backward chains, respectively. We have reached this point because one or more items of the last column of `x.` and the first column of `y.` match. We want to keep only those rows containing these matching items. The first two lines remove from `x.` and `y.` all the rows that don't have matching values in them. I'll invent an `x.` and a `y.` and go slowly through the steps that lead to the desired result.

Here are the two:

```
   x.
200 300 400
  0   1 130
  2   3 120
  4   5 130
  6   7 120
  8   9 130
500 600 700

   y.
500  600  700  800
130    2    1    0
120    5    4    3
120    8    7    6
120   11   10    9
120   14   13   12
130   17   16   15
900 1000 1100 1200
```

Only five rows of `x.` and six of `y.` match. First we remove the rows of `x.` that don't end in one of the matching values:

```
   ] x.=. x.#~ ({:"1 x.) e. {."1 y.
0 1 130
2 3 120
4 5 130
6 7 120
8 9 130
```

And similarly, remove the rows of `y.` that don't begin with one of the matching values.

```
   ] y.=. y.#~ ({."1 y.) e. {:"1 x.
130  2  1  0
120  5  4  3
120  8  7  6
120 11 10  9
120 14 13 12
130 17 16 15
```

Now we have to maneuver to get the rows of `x.` ending in 120 in line with those of `y.` beginning with 120, and similarly for 130. First we compare for equality the tail of each row of `x.` with the head of each row of `y.`:

```
   ({:"1 x.)=/{."1 y.
1 0 0 0 0 1
0 1 1 1 1 0
1 0 0 0 0 1
0 1 1 1 1 0
1 0 0 0 0 1
```

This is ravelled and the indices of 1s found:

```
   I.,({:"1 x.)=/{."1 y.
0 5 7 8 9 10 12 17 19 20 21 22 24 29
```

We convert this into their base `#y.` representation.

```
   (0,#y.)#:I.,({:"1 x.)=/{."1 y.
0 0
0 5
1 1
1 2
1 3
1 4
2 0
2 5
3 1
3 2
3 3
```

```
3 4
4 0
4 5
```

It's another Magical Matrix [1]. The first column is used to select rows from `x.` and the second to select rows from `y..`

Use the second column to select rows from `y.` in the right quantity and in the right order:

```
   ({:"1 i){y.
130  2  1  0
130 17 16 15
120  5  4  3
120  8  7  6
120 11 10  9
120 14 13 12
130  2  1  0
130 17 16 15
120  5  4  3
120  8  7  6
120 11 10  9
120 14 13 12
130  2  1  0
130 17 16 15
```

and use the first column to select rows of `x.`, at the same time removing the last column; it merely repeats the first column of `y..`

```
   (({."1 i){}:"1 x.)
0 1
0 1
2 3
2 3
2 3
2 3
4 5
4 5
6 7
6 7
6 7
6 7
8 9
8 9
```

Lastly, stitch these together, and we have the desired result:

```
   (({."1 i){}:"1 x.) ,. ({:"1 i){y. [ i=. (0,#y.)#:I.,({:"1
x.)=/{."1 y.
0 1 130  2  1  0
0 1 130 17 16 15
2 3 120  5  4  3
2 3 120  8  7  6
2 3 120 11 10  9
2 3 120 14 13 12
4 5 130  2  1  0
4 5 130 17 16 15
6 7 120  5  4  3
6 7 120  8  7  6
6 7 120 11 10  9
6 7 120 14 13 12
8 9 130  2  1  0
8 9 130 17 16 15
```

And we've matched the three 130s in the last column of `x.` with the two 130s in the first column of `y.`, giving six rows; and matched the two 120s in the last column of `x.` with the four 120s in the first column of `y.`, giving eight rows, which makes fourteen rows altogether. This contrived example shows a solution where there are fourteen different ladders with six links each, giving the links we can use to select the words that make Word Ladders.

Acknowledgements:

R. E. Boss provided a workable joining function on the same day that I sent a message to the J forum asking for one. Norman Thomson gave me many ideas about finding paths in graphs. Roger Hui took my ugly ducklings and turned them into beautiful swans, and, by jumping to a conclusion, made it possible for me to understand better the structure of the problem.

Reference

[1] McDonnell, Eugene, "The Magical Matrix", *Vector 20* No. 2, October 2003, pp 122-6

Dyalog Next Generation: Un-named Window Refs

by Peter-Michael Hager (hager@dortmund.net)

With version 9, besides variables, functions and operators, Dyalog had introduced a new data type: the object reference, known as "ref".

Refs are pointers to namespaces and can be assigned, compared and formatted. So they behave in a very similar way to variables.

The result of formatting a ref is the name of the object, where the ref points to. If the result is identical with the full qualified name of the ref, the ref represents the object itself, a self reference.

Refs need not be references to named objects. Using ⎕NS'' refs to un-named objects can be generated. Un-named objects can never have a self reference. So, the formatted result of such an un-named ref looks like this:

```
      ⍕oRef←⎕NS''
#.[Namespace]
```

Instead of the object name, which cannot exist for un-named objects, the formatted result of an un-named ref is the path to its class (here: Namespace), distinguished by being enclosed in square brackets.

The structure type of object refs is native scalar and they may be composed to vectors, matrices, etc., among other structures. In accordance with the rules of array processing in APL, structures of refs are executed in parallel too:

```
      os←⎕NS¨4⍴⊂''
      os[1].F←+
      os[2].F←-
      os[3].F←×
      os[4].F←÷
      8 os.F ⍳4
9 6 24 2

      os[1 3].(F/⍳10)
55 3628800
```

Beginning with version 10 of Dyalog APL, now these un-named refs also act on Windows' GUI classes, by an appended root method, realized as a dynamic function:

```
#.CreateObject←{α←⍎↑⎕NSI ⋄ ω{ω{α{α}α.##.⎕EX'o'}'o'ω.##.⎕WC α}α.o←α.⎕NS''}
```

Syntactically the right hand side of #.CreateObject is identical to that from ⎕WC. The result from `#.CreateObject`, however, is an un-named object ref, or you might call it an instance of the class given by the Type property.

If the object ref does not get assigned, or when its assignment is deleted, the object instance will disappear.

Optionally a parent object ref might be given in the left argument:

```
      )CS #.Samples
#.Samples
      oForm  ←      #.CreateObject'Form'
      oButton←oForm #.CreateObject'Button'
      ⍕oForm
#.Samples.[Form]
      ⍕oButton
#.Samples.[Form].[Button]
      osObject←oForm oButton
      ⍕osObject
 #.Samples.[Form]  #.Samples.[Form].[Button]
```

In contrast to the conventional concept, un-named window refs have several advantages:

- The interpreter no longer needs to analyse textual character strings to find the path to an object, and does not have to administer a name tree, both increasing its performance.
- The graphical layout (skin), the underlying functionality (callbacks), and the localization (language), can be separated in a program more easily:

```
[0]   Localize Parms;osObject;Language
[1]   osObject Language←Parms
[2]   :Select Language
[3]   :Case 'ENG'
[4]       osObject.Caption←'Sample Form' 'Send'
[5]   :Case 'GER'
[6]       osObject.Caption←'Beispiel-Form' 'Senden'
[7]   :EndSelect

      Localize osObject 'GER'
```

- Any desired number of instances, as necessary for MDIs or services, can be generated easily:

```
OneForm;oForm;oCloseFile;oShowEdit;oTestEdit;oTestButton
oForm oCloseFile oShowEdit oTestEdit oTestButton←1 1 0 1 1 1/CreateForm 1
oCloseFile.onSelect←'_onCloseFileSelect'(,oForm)
oShowEdit.onSelect←'_onShowEditSelect'
oTestEdit.onSelect←'_onTestEditSelect'(oForm oShowEdit)
oTestButton.onSelect←'_onTestButtonSelect'(oForm oShowEdit)
#.Wait

SixForms;osForm;osCloseFile;osCloseAllFile;osShowEdit;osTestEdit;osTestButton
osForm osCloseFile osCloseAllFile osShowEdit osTestEdit osTestButton←
      ⊂[1]⊃CreateForm¨s6
osCloseFile.onSelect←(⊂'_onCloseFileSelect')∘,∘⊂∘,¨osForm
osCloseAllFile.onSelect←1
osShowEdit.onSelect←⊂'_onShowEditSelect'
osTestEdit.onSelect←(⊂'_onTestEditSelect')∘,∘⊂∘,¨osForm,¨osShowEdit
osTestButton.onSelect←(⊂'_onTestButtonSelect')∘,∘⊂∘,¨osForm,¨osShowEdit
#.Wait
```

The only disadvantage at the moment is the incomplete integration. So, the CreateObject method should exist in every class (might be called CreateChild?), and objects like `FontObj`, `IconObj`, `SplitObj`, should be assignable as refs. The current flaw of the monadic `⎕NS ''` needs to be removed, to make it possible to duplicate instances simply by:

```
osObjects←⎕NS¨6⍴⊂osObject
```

The introduction of a method like Instantiate, instead of `⎕NS`, and a property like Prototype, similar to Visible and Active set to zero, which would get reset by calling Instantiate, would be nice.

Even more, a fill-element (will be a namespace of course) for refs, like blank or 0 for variables, will be mandatory.

Anyway, even with the current implementation, Dyalog has gone a great step towards object abstraction, which might allow APL to become a better object language than all these attempts like the plus-pluses.

The APL2-Java™ Interface

by David Liebtag
(APL Products and Services, IBM Silicon Valley Laboratory)

The IBM APL group is working on an exciting new facility: the APL2-Java™ interface. Through this facility, you will be able to call Java programs and have those programs call back to APL2. The new facility will be available in Workstation APL2 Service Level 4.

Calling Java from APL2

The APL2-Java interface includes a new associated processor 14 which is used to call Java from APL2. A processor 14 association looks like this:

```
('Class' 'Signature') 14 ⎕NA 'Surrogate Member'
```

Where:

- 'Class' is the name of a Java class or an instance returned by a constructor
- 'Signature' is a description of a Java field's datatype or a Java method's arguments and results.
- Member is the name of the Java field or method
- Surrogate is the name to associate with the member. (A surrogate is required if the member name is not a valid APL2 object name. For example, constructors are always named <init> and so always require a surrogate.)

Note: For those readers who do not know Java, please see the Java Language Overview at the end of this article.

Here is an example of using static members of a Java class named Sample.

```
('Sample' 'I') 14 ⎕NA 'StaticField'
1
StaticField ← 24
('Sample' '(I)I') 14 ⎕NA 'StaticMethod'
1
Added ← StaticMethod 47
```

Here is an example of using an instance of the Sample Java class

```
('Sample' '(I)V') 14 ⎕NA 'Constructor <init>'
1
Object ← Constructor 999
(Object '(I)I') 14 ⎕NA 'InstanceMethod'
1
Result ← InstanceMethod 456
```

Processor 14 includes several built-in external functions that are used to manage the APL2-Java interface. For example, the DeleteLocalRef function is used to delete objects created through Processor 14:

```
3 14 ⎕NA 'DeleteLocalRef'
DeleteLocalRef Object
```

Calling APL2 from Java

The APL2-Java interface includes several Java classes that enable Java programs to call APL2. Using these classes, Java programs can start APL2 interpreters, associate and create objects in workspaces, execute APL2 expressions, and call APL2 functions.

The Apl2interp class includes constructors that are used to start new APL2 interpreters. For example, the following Java code starts a slave APL2 interpreter:

```
Apl2interp Slave = new Apl2interp() ;
```

The Apl2interp class also contains methods for associating names. For example, if you had packaged the GRAPHPAK workspace, you could use the slave interpreter to associate the names PLOT and VIEW.

```
Slave.Associate("GRAPHPAK","PLOT") ;
Slave.Associate("GRAPHPAK","VIEW") ;
```

The Apl2object class is used to create and manipulate APL2 workspace objects. The following code creates an object in the slave interpreter's workspace from a Java array:

```
int[] Array = {2,34,6,28,19} ;
Apl2object PlotData = new Apl2object(Slave,Array) ;
```

The Apl2interp class also contains methods for executing APL2 expressions. This code executes the PLOT and VIEW functions:

```
Slave.Execute("PLOT",PlotData);
Slave.Execute("VIEW") ;
```

Calling Back to APL2

Java programs called through Processor 14 can also call back to the APL2 interpreter. The Processor 14 built-in function GetApl2interp retrieves the instance of the Apl2interp class associated with the current APL2 interpreter. For example, the following code passes the result of GetApl2interp to a Java method:

```
3 14 ⎕NA 'GetApl2interp'
Signature ← '(Lcom/ibm/apl2/Apl2interp;)V'
('Sample' Signature) 14 ⎕NA 'CallBack'
CallBack GetApl2interp
```

The CallBack method might look like this:

```
public void CallBack(Apl2interp Apl2) {
   Apl2object Message = new Apl2object(Apl2,"Hello") ;
   Apl2.Assign("VAR",Message) ;
   Return ;
   }
```

The example assigns the character vector 'Hello' to a variable named VAR.

Summary

The IBM APL Products and Services group is working to transform APL2. Rather than the traditional stand-alone application environment, the new APL2 will fit seamlessly in modern programming environments. Using the new APL2-Java interface, APL2 application developers will be able to leverage the vast wealth of utility packages available through Java. In addition, developers can call APL2 applications from Java code and deploy their applications using mainstream tools such as WebSphere Application Server.

Java Language Overview

Java programs are composed of Java classes. Each class performs a service such as file IO, database access, or user interface services. Some classes are built-in to Java; some are built by application developers.

A Java class is similar to an APL workspace. Like a workspace, a class resides in a single file. Also like a workspace, a class can contain variables and functions. And

also like a workspace, variables can be global or local. In addition, functions can also be local or global.

Here is a sample class:

```
public class Sample
{
   public static int StaticField = 0 ;
   public static int StaticMethod(int Argument) {
      return Argument + StaticField ;
   }

   public int InstanceField ;

   public Sample(int InitialValue) {
      this.InstanceField = InitialValue ;
      return ;
   }

   public int InstanceMethod(int Argument) {
      return Argument + InstanceField ;
   }
}
```

The Java designers decided to use different words for things than we use in the APL community. In Java,

- Public means global
- Private means local
- Field means variable
- Method means function
- Member means either a field or a method.

The first line of the class definition supplies the name of the class, in this case Sample. Everything within the curly braces constitutes the class's definition.

The keyword public indicates that a field or method is accessible from outside the class.

The first two members are declared static. This means that they exist as soon as the class is loaded and can always be used. They correspond to global variables and functions in an APL workspace.

The next member, InstanceField, is not declared static; this means that it is an "instance" member and can only be used in the contact of an object. To create an object, you must call a special method called a constructor. Constructors have the same name as the class; the fourth item is a constructor. When you call the Sample constructor, an instance of the Sample class is created. Once you have an instance of the class, you can use the instance, also called an object, to access instance fields and methods.

To illustrate, here is some Java code that uses the Sample class:

```
//* Set the static field
Sample.StaticField = 24 ;

//* Use the static method
int Added = Sample.StaticMethod(47) ;

//* Create an instance of the Sample class, a Sample object
Sample Object = new Sample(999) ;

//* Call the instance method
int Sum = Object.InstanceMethod(456) ;
```

Notice for static members, you use the name of the class followed by a period and the member name. For instance members, you use the name of a field containing a reference to the instance followed by a period and the member name.

Trademark attribution statements

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

IBM, APL2, and WebSphere are trademarks of International Business Machines Corporation in the United States, or other countries, or both.

An APL to C++ Translator

by Dr. James A. Brown, Chairman and CEO SmartArrays, Inc.

Abstract

What's the hardest part about writing an APL Compiler? Is it the parsing of the language? No. APL syntax is relatively simple and is easy to parse. Is it the understanding of the intended operations? Not really, APL operations are high level and well understood. The real difficulty is that the target languages, typically C or C++, are so low level. The basic operations of APL do not exist in the target languages.

SmartArrays is a class library for C++ (and other languages) that provides a full set of APL-like array operations. This paper describes how C++ enhanced by SmartArrays can be a target for APL translation.

Introduction

There have been many efforts to write translators that convert APL2 to a compiled language. This writer managed an APL Compiler effort at IBM Yorktown for a time. SmartArrays provides the basic array operations and data of APL to the compiled languages easing the task of translating APL to these languages.

What is SmartArrays?

SmartArrays, from the programmer's point of view, is a class library that can be used from C++, C#, and Java to provide to these languages APL-like multidimensional nested arrays and the functions that operate on them. The primary array object is called an 'SmArray' SmartArrays methods operate on SmArrays and return an SmArray as a result just like APL operations. For example, if you wanted to write in C++ some code that did this APL2 operation:

```
B←⍋+/⌽A
```

You could write:

```
B = A.reverse().reduce (Sm::plus).gradeUp();
```

Virtually all of the APL primitive functions and operators that are not spelled with a Quad in their name are available as methods in SmartArrays. There are also

many additional array methods in SmartArrays some of which implement idiomatic expressions (like **rank()** and **unique()**) while others are functions not available as primitives in APL (like XML, statistical, and sparse cube methods). These additional methods are not produced by the translator but might be used when fine tuning the program produced by the translator.

What is the SmartArrays APL translator?

The SmartArrays APL translator is an APL program that converts APL functions to C++ programs. It can be applied to a single APL function or to the main function of an application in which case it will dynamically determine what sub-functions are used and translate them also.

You translate an entire workspace of functions with a statement like the following where "MAIN" is the name of the main program of the application:

```
SmATranslateWs 'MAIN'
```

You can also translate a single defined function like this:

```
SmATranslateFn 'Funname'
```

You would not normally translate a single line of APL except when experimenting with the translator to see what it does with various expressions. You'll see many examples of this in this paper as a way to present various examples.

What can the SmartArrays translator translate?

Virtually all of the APL primitive functions and operators that are not spelled with a Quad in their name can be translated. In addition, Quad for output and Quad IO for origin control are translated.

What cannot be translated?

The SmartArrays APL translator is designed to be used for translating algorithms involving array computing. It would not normally be used with code that implements a user interface. It would not normally be used for code that manages files. (Although SmartArrays does contain the ability to treat a file as an array.) In general, system functions and system variables cannot be translated. The APL 'execute' function and defined operators cannot be translated. Operators on defined functions are not supported. Complex numbers and functions on complex

numbers cannot be translated. Namespaces are not supported. Support for Global variables is limited.

APL does branching by using line numbers with labels as named constants as a syntactic helper for defining line numbers. The host languages for SmartArrays and the translator treat labels as names not numbers. Therefore, the translator must be able to determine the names of labels being selected and convert expressions to C++ using these names. Therefore, not all forms of branching can be translated.

Some Simple Examples

Here is a sequence of scalar functions. Notice that because the C++ arithmetic operations are overloaded for arguments that are SmartArrays, the resulting C++ code is very similar to the APL expression:

```
      SmA_TranslateLine 'X←A+B×÷C'
X = (A+(B*C.recip()));
```

Expressions that are not in the C++ vocabulary are translated using the normal object-method application syntax of C++:

```
      SmA_TranslateLine 'X←A⍴B×⌊C'
X = A.reshape((B*C.floor()));
```

Selective assignment expressions are translated using the SmartArrays "Into" methods:

```
      ⊃SmA_TranslateLine 'V[I]← X'
V.indexInto(X,I);
```

Branching expressions are translated into C++ 'goto' statements:

```
      ⊃SmA_TranslateLine '→(A<B)/L1'
if ((A<B).getInt() )
    goto L1;
```

Index Origin

The compiled languages do not have the concept of index origin and always use zero to indicate the first item of a collection. Thus, APL programs written to use zero-origin are easier to translate. To translate a function that uses one-origin, values that select items may need to be adjusted. This can be done automatically

for constant expressions. Here are some examples of origin dependent statements in a one-origin workspace where the origin can be adjusted automatically:

```
      ⊃SmA_TranslateLine 'A←X[1]'
   A = X[0];

      ⊃SmA_TranslateLine 'A←X[⍳3]'
   A = X[SmArray::scalar(3).sequence()];
```

Here are some examples of origin dependent statements where the translator has to ask for help with origin dependency:

In the first case, the programmer would need to know how the variable 'I' was defined to determine if it should be adjusted or not. In the second case, the fact that the index is a subset of a generated interval, it does not need to be adjusted.

```
      ⊃SmA_TranslateLine 'A←X[I]'
Adjust 'I' for origin? (Y/N) y
    A = X[I-1];

      ⊃SmA_TranslateLine 'A←X[I/⍳⍴X]'
Adjust 'I.replicate(X.shape().sequence())' for origin? (Y/N) n
    A = X[I.replicate(X.shape().sequence())];
```

The first version of the translator does not do any function or statement analysis beyond a single primitive. Enhancements to the translator could increase the number of cases that can be automatically translated. Settings of local origin are detected and honoured.

Here is a the translation of a trivial function that modifies a local origin:

```
      ⊃SmA_TranslateFn'ORGTest'
  [0] Z←ORGTest R;⎕IO
  [1] ⎕IO←1
  [2] Z←R[I]
Adjust 'I' for origin? (Y/N) y
  [3]
  [4] ⎕IO←0
  [5] Z←R[I]

SmArray ORGTest (SmArray R){
  try {

    SmArray Z;
    Z = R[I-1];

    Z = R[I];
      return Z;

  } catch( SmException e ){
    std::cout << "ORGTest(): " << e.toString() << std::endl;
    throw;
  }
return 0;
}
```

Since there is no information about the value of **I**, when the translator sees **I** being used as an index, it has to ask if it should be adjusted. However, the exact same line encountered after the origin is set to zero does not require any interaction. Notice how the two translations of the index expression differ.

Translator Directives

In an APL program, all data is an array be it a complicated multi-dimensional nested array or a simple scalar. By default, when you translate a program from APL to C++, all operations are converted to SmartArrays methods and all data is converted to SmArrays. Here's an example of a program that might have been written before the addition of control structures in APL that uses a scalar integer to control a loop:

```
[0]    Z←MAX CONTROL XS;I;⎕IO
[1]     ⎕IO←0
[2]     I←¯1
[3]    L1:I←I+1
[4]     XS[I]←XX[I]+REFIT PR[I]
[5]     →(I<MAX)/L1
[6]     Z←+/XS
```

By default when this is translated to C++, you get a program that computes entirely with SmartArrays methods applied to SmArrays:

```
SmArray CONTROL (SmArray MAX, SmArray XS){
  try {

    SmArray Z;
    SmArray I;
    I = SmArray::scalar(-1);
  L1:
    I = (I+1);
    XS.indexInto((XX[I]+REFIT(PR[I])),I);
    if ((I<MAX).getInt() )
        goto L1;
    Z = XS.reduceDescending(Sm::plus);
      return Z;

  } catch( SmException e ){
    std::cout << "CONTROL(): " << e.toString() << std::endl;
    throw;
  }
return 0;
}
```

This is an inefficient way to loop in APL and it's an inefficient way to loop in SmartArrays. If you were writing a SmartArrays program by hand, you would normally freely intermix SmartArrays data and data of the host language and you would use SmartArrays methods and native methods as appropriate.

The translator supports a directive that may be used to declare that a set of names in your program should be treated as scalar integers. This directive is added to your APL function as a comment. Here is the directive that is useful in the CONTROL function:

```
ASmA ScalarInt I MAX
```

This directive can be anywhere in the program. Here is the program produced

```
SmArray CONTROL (long MAX, SmArray XS){
  try {

    SmArray Z;
    long I;
    // SmA ScalarInt I MAX
    I = -1;
  L1:
    I = I+1;
    XS.indexInto((XX[I]+REFIT(PR[I])),I);
    if (I<MAX)
        goto L1;
    Z = XS.reduceDescending(Sm::plus);
      return Z;

  } catch( SmException e ){
    std::cout << "CONTROL(): " << e.toString() << std::endl;
    throw;
  }
return 0;
}
```

Note that the looping is now done using scalar integer operations of the native language.

Conclusion

This paper describes Release 1 of the SmartArrays translator. It has already been used to convert significant algorithms into C++. Interestingly, the company that did these translations did not do it so they could move their program development from APL to C++ but rather so they could continue to use APL for their sophisticated and constantly fine-tuned algorithms but still deliver applications in C++.

Many of the restrictions present in this first release could be removed in future versions. This writer implemented idiom recognition in IBM APL2 so that common idioms execute with the performance of a single primitive. This kind of analysis could be added to the translator. More sophisticated tracking of expressions in a one-origin workspace could lead to more completely automatic origin adjustment.

True global variables could be implemented by wrapping the translation of an APL workspace in a C++ class definition.

Versions of the translator that target C# or Java could be produced (although this is largely already supported). Versions of the translator tailored to APL2000 or Dyalog APL could be produced (although, again, this is largely already supported). Namespaces could be supported.

The SmartArrays translator can be used by companies that need to move their development to a compiled language. It can also be used by companies that want to stay in an APL development environment but deliver applications in the compiled languages. For these companies, APL could be the design language for a whole new range of applications.

A Strategy for Compiling APL for the .Net Framework

by Adrian Smith (adrian@causeway.co.uk)

Abstract

The APL literature has many papers on APL compilation, mostly dating from before 1982. What was possible then should be more possible now, given the strong vector support and excellent garbage-collection in the .Net *Common Language Runtime* (CLR). This paper outlines the approach being taken by Causeway to ship the *RainPro* and *NewLeaf* engines as pure .Net DLLs. This involves some hard choices on what subset of APL is truly essential. The rationale for these choices is discussed, and various conversions from Dyalog APL to C# are illustrated.

Motivation

Simple survival. If Causeway is to continue to ship 'possibly the best graphics engine in the world' to the next generation of .Net users, it has to be 'pure managed code' and it should be fast, lightweight and ideally a single DLL. This means that we cannot continue to bundle a complete runtime APL interpreter with our distribution. We cannot rewrite in C# – the only specification we have is the existing APL codebase. Our only path forward is to attempt to regenerate this APL treasure-house into C# source and compile that.

We also believe that to move wholesale to C# would freeze innovation and development in its tracks. APL is still the most productive environment for playing with ideas, experimenting with data and prototyping new solutions. We want to keep our development in APL; we just want to ship the results in .Net.

Literature Search

This may be an old-fashioned approach, but I started by tracking down every paper I could find on APL and compilation. There is a goldmine in old conference proceedings from 1978 to around 1982, then along came APL2 and it just ends there. Maybe that tells you something about APL2?

The overwhelming conclusion was that with 'very few restrictions' VS APL is perfectly possible to compile. Most of the restrictions are either plain common sense (clearly you can't use *execute* or *quadFX* to change the meaning of a token

dynamically) or accepted as good habits by the vast majority of modern APLers (don't re-use locals with different data types, don't look up the stack for data localised in your caller, avoid fancy nested structures if a simple array will do the job, ...) so no great issue there. In common with all APLers, I break my own rules fairly frequently, but I can face the pain of tidying up my APL code to comply with strictures like these. It will probably improve as a result.

The other valuable resources were the survey papers (for example from STSC) showing the usage frequency of the primitives, the inner and outer products, and the rank of data most commonly encountered by the simple primitives like plus and times. This last was quite surprising, but very encouraging. Around 99.6% of ALL expressions were found to operate on arrays of rank 0,1,2 (over a 2-week period of typical usage of the STSC timesharing service). Conclusion – if we write an engine that does scalars, vectors and matrices, it should cover 99.6% of the 'typical' APL system. I can live with that.

A similar survey found the average length of a vector to be 16. If this is typical of our code, then we might find that eliminating interpreter overhead might actually give us a considerable speed-up. An APL engine can only compete for speed with compiled code when the average length of a vector gets up into the thousands.

Spike Test – Eratosthenes yet again

The classic prime-number sieve is my favourite testbed for this kind of experiment. It loops a bit, it hits vectors (and hence memory-management) pretty hard, and you don't need many primitives to make it run. Here is the APL code:

```
      ∇ primes←Eratos max;candidates;check
[1]     ⍝ Returns primes up to n by the sieve of Eratosthenes
[2]     primes←1
[3]     candidates←1↓⍳max
[4]
[5]     :While 0<⍴candidates
[6]       primes←primes,check←candidates[1]
[7]       candidates←(0<check|candidates)/candidates
[8]     :End
      ∇

      ⍴Eratos 100000
9593
```

... which runs in around 10sec in Dyalog APL on my 400MHz laptop. This is a fairly hard target to beat, as it loops 9593 times, but must repeatedly compress a reasonably long vector of possible primes each time around the loop.

At the other end of the scale, we have a hand-coded C# version:

```
using System;
using System.Collections;

class MainClass
{
   public static void Main(string[] args)
   {
      int val = 100000;
      if(args.Length > 0) val = Int32.Parse(args[0]);
      Console.WriteLine("Start with "+val+" ...");
      ArrayList e = Eratos(val);
      Console.WriteLine("Done - " + e.Count);
   }

   private static ArrayList Eratos(int max){
      ArrayList primes = new ArrayList();
      primes.Add(1);
      int[] candidates = new int[max - 1];
      for(int i = 0; i < max - 1; i++) candidates[i] = i + 2;

      while(candidates.Length > 0){
         int check = candidates[0];
         primes.Add(check);
         bool[] bv = new bool[candidates.Length];
         for(int i = 0; i < candidates.Length; i++) bv[i] = (candidates[i] % check) > 0;
         candidates = Compress(candidates, bv);
      }

      return primes;
   }

   private static int[] Compress(int[] from, bool[] by){
      int size = 0;
      foreach(bool b in by) if(b) size++;
      int[] res = new int[size];
      int resp = 0;
      for(int i = 0; i < by.Length; i++) if(by[i]) res[resp++] = from[i];
      return res;
   }
}
```

This runs in just over 8sec on the same hardware, and is a fairly good facsimile of the APL algorithm, written by Richard Smith in about 20 minutes. It proves a number of things:

1. The C# runtime is definitely not a dog. Experiences of Java made us wary of any environment which compiles to a virtual machine. It would appear that the Microsoft guys have not hit us with a speed penalty here.

2. The garbage collection works really well. That *candidates* vector gets re-assigned and discarded 9593 times and there is no visible hit on the memory usage of the machine while this is running. Good.

3. The code required to implement *compress* is pretty straightforward. OK, this just does the vector case, and only for integers, but you can see that the C# array capabilities are not too stupid. Arrays are fully dynamic, and indexing works as you expect. They even know their own length.

4. That *while* structure looks suspiciously similar to the APL version. Maybe (just maybe) the modern APL control structures map across almost unchanged. That would be a major bonus.

Conclusion - this is worth a go. So how do we set about transforming our APL source-code into the C# version automatically? Time to make some more choices.

Route-1 - the easy way

Imagine a library of APL primitives where everything simply takes *object* and returns *object*. Imagine a memory-manager class which can store data for you, and even imitate the APL scoping rules. Well, we did more than imagine it; in the cool of the evening in Vancouver (Causeway did *SVG-Open* last year) we coded enough of it to run *Eratos*. Which it did, like a dog. Like a 2min 4sec dog, in fact. The C# code looks fine, but the snag with this approach is that every primitive has to run through a block of code to find out what kind of thing it was passed. Then it must coerce one or both arguments to the most general type (say *double[]* for most of the arithmetic stuff) before doing whatever it does, before wrapping the result up as *object* ready to return it. We knew it would be slow, but not *this* slow. Back to the drawing board.

Route-2 - the *SmartArrays* (and *Rowan*) way

Both Jim Brown and Richard Smith take the approach of making the Java/C# programmer declare a specific data-type (Jim's is an *smArray*) which is a little structure just like the typical APL array of mainframe days. Effectively this describes the data content for you, saving nearly all the overhead incurred by Route-1. *Rowan* runs *Eratos* in about 18sec, so this is definitely a tenable approach. We chose not to do it, because we wanted to generate C# that looked like the C# a 'real' programmer would write. We also took note of just how much scalar code there is in the average chunk of commercial APL, and felt that on balance our primitives should (for example) be able to add an *int* to an *int[]* without the extra fuss of declaring our own special data-type.

Route-3 - use native C# data-types everywhere

The spike-test for this one took all of a morning to write, and it ran *Eratos* in 11sec. The downside is that you need lots and lots of overloads for all the primitive functions - at the last count *AE.Times* managed 63 variants, working upwards

from *Times(int,int)* and including *Times(int,bool[])* and so on. Unfortunately, C# regards rank-2 and rank-3 arrays as different types, so we took a hard decision and stopped at rank-2, just to keep everyone sane. One or two really oddball functions (like *QuadGets*) do take *object* and try to figure it out at runtime, but we came to the view that in general, the benefit of strong typing was worth the pain.

Of course, most of the pain comes at the APL end – this approach actually makes the function-library much simpler to write (given some suitable template-expansion code) and the C# compiler takes almost all the decisions at compile time. It also eliminates nearly all run-time errors – if you can get your converted APL past the compiler, it will almost certainly run correctly.

The hard part is moved over to the code-convertor, which must be able to deduce the correct data-type for all the local variables *lexically*. I did toy with the idea of infecting the target functions with a suitable analyser, running the application a few times, and then patching the original APL code with declarative comments. It felt like a kludge, but it did at least reassure me that I was fairly reliable in avoiding local-variable re-use. Much better to do it right.

Figuring out Data-types at Parse Time

Well, the C# compiler can do it (how else does it know to call the correct overload?) so surely we can do the same trick? Let's start with something really obvious ...

```
      cc.chk 'a←⍳12'

       a      ←      ⍳      12
MARK   NOUN   ASGN   VERB   NOUN
a = AE.Range(12);   int[]
```

A brief word of explanation. I have a `cc` namespace which does all the hard stuff. The first stage is called **Word Formation** and this is followed by **Parsing**. All `chk` does is run these back to back with ⎕← in between so you can see what the word-formation saw. I copied Roger Hui's parse-table hook, line and sinker from *Vector* Vol.9 No.4 (p.92) so I use the J terminology throughout.

The two-element result from `cc.chk` gives me the converted expression, and the data-type of the final result. It takes an optional left argument which I can use to tell it the types of any previously assigned names.

For example:

```
      ('double[]' 'local') cc.chk 'a←⍳local'
       a      ←      ⍳      local
MARK  NOUN   ASGN   VERB   NOUN
TYPE - double[] not supported by AE.Range
a = AE.Range(local);  ????
```

Not unreasonably, this gets upset about being asked to run ⍳ on a floating-point vector. It has a go at converting the statement, but returns ???? as the resulting datatype, as this is undetermined at this point. If we were to run:

```
      'int' 'local' cc.chk 'a←⍳local'
       a      ←      ⍳      local
MARK  NOUN   ASGN   VERB   NOUN
a = AE.Range(local);  int[]
```

... then everything is fine. So how about:

```
      'int' 'local' cc.chk 'a←12⍴⍳local'
       a      ←      12     ⍴      ⍳      local
MARK  NOUN   ASGN   NOUN   VERB   VERB   NOUN
a = AE.Reshape(12,AE.Range(local));  int[]
```

... and so on. It looks tricky, but actually it is remarkably easy for the parser to match up the incoming data-types for each primitive with the (unique) combination of data-types supported by the corresponding primitive in the library, and hence unambiguously establish the return type. We just keep rolling along, adding parentheses as we go:

```
      'int' 'local' cc.chk 'a←2.5×12⍴⍳local'
       a      ←      2.5    ×      12     ⍴      ⍳      local
MARK  NOUN   ASGN   NOUN   VERB   NOUN   VERB   VERB   NOUN
a = AE.Times(2.5,AE.Reshape(12,AE.Range(local)));  double[]
```

... and when we hit an assignment, we store away the incontrovertible fact that a is now (beyond a shadow of a doubt) an array of type *double[]*. If a subsequent line should try to assign something else, we splatter a message all over the session.

Which just leaves:

```
     cc.chk 'a←1 0 1'
      a      ←      1 0 1
MARK  NOUN   ASGN   NOUN
a = new int[] {1,0,1};  int[]
```

... and a few things like it. Is this an array of type *bool[]* or *int[]* or even *double[]*? There is no way we can know, so I am afraid you just have to declare these.

Fortunately, this sort of thing doesn't come up too often in real code and when it does, the convertor can take the hint:

```
      ('double[]' 'a') cc.chk 'a←1 0 1'
         a      ←      1 0 1
MARK   NOUN   ASGN   NOUN
a = new double[] {1,0,1};  double[]

      ('bool[]' 'a') cc.chk 'a←1 0 1'
         a      ←      1 0 1
MARK   NOUN   ASGN   NOUN
a = new bool[] {true,false,true};  bool[]
```

and poor *zilde* gets some serious abuse in this case:

```
      ('int[]' 'a') cc.chk 'a←⍬'
         a      ←      ⍬
MARK   NOUN   ASGN   NOUN
a = new int[0];  int[]

      ('bool[][]' 'a') cc.chk 'a←⍬'
         a      ←      ⍬
MARK   NOUN   ASGN   NOUN
a = new bool[0][];  bool[][]
```

Basically, the default assumption is *int* or *int[]* if there is nothing to go on, otherwise any constant vector will be formatted up to match the pre-set type of the name to the left of the assignment. Which just leaves:

```
      cc.chk 'a←∨/1 0 1'
         a      ←      ∨/     1 0 1
MARK   NOUN   ASGN   VERB   NOUN
TYPE - int[] not supported by AE.OrReduce
```

... where the default assumption is clearly wrong, and the error spurious. Again, this hardly ever happens in real application code, but the way out (until I can think of something better) is to define a pair of globals `true` and `false` in your workspace, and recode the above as:

```
      cc.chk 'a←∨/true false true'
         a      ←      ∨/     ⊣true false true⊢
MARK   NOUN   ASGN   VERB   NOUN
a = AE.OrReduce(new bool[]{true,false,true});  bool
```

... which stops up that hole for us. If (for some very obscure reason) we wanted it treated as *double[]* here (rather than *int[]*) the parser has one more trick up its sleeve.

Consider:

```
      cc.chk 'a←+1 0 1'
         a      ←      +      1 0 1
MARK   NOUN   ASGN   VERB   NOUN
a = AE.Promote(new int[] {1,0,1});  double[]
```

Purists may scream now. What we chose to do is to implement the C# version of monadic *plus* as 'leave the value unchanged, but increment the data-type' so an extreme case would be:

```
      cc.chk 'a←+++true false'
         a      ←      +      +      +      ⊣true false⊢
MARK   NOUN   ASGN   VERB   VERB   NOUN
a = AE.Promote(AE.Promote(new bool[]{true,false}));  double[]
```

Of course you can do the usual stuff with `floor` and `1=` to demote to integer and boolean respectively. Note (in passing) that strands are treated as NOUNs here, but get flagged by the left and right tack, just to be sure. There is a reason for this, to do with multiple arguments to functions. We will come to this later.

Stocktaking - what do we have and what do we need?

Well, it looks pretty plausible that we can convert an APL expression to a C# expression which will do the exact same job, as long as the data-types are knowable in advance. There will be a few bear-traps like:

```
'int[]' 'qq' cc.chk 'qq/⍨←msk'
      qq     /      ⍨      ←      msk
MARK   NOUN   VERB   ADVB   ASGN   NOUN
qq = AE.Compress(msk,qq);  int[]
```

... but these all prove to be a lot easier than you might think. Once you have decided to use a Hui-style parse-table, then fancy syntax is just adding patterns to the table, and all the Dyalog extensions fall out pretty easily.

So - we can convert an expression. The next challenge is to convert a function, which adds a few new challenges, starting with branching and control-structures. Let's tackle these next.

GoTo Considered (mostly) harmless

I very, very rarely use branches and labels in code I write today, but it seemed stupid to force a rewrite of old code just for the sake of it, when C# supports labels and *Goto* quite happily.

Here is my testbed:

```
      ∇ msg←branches fred
[1]    ⍝ Assorted branch styles
[2]    ⍝:Public
[3]    ⍝: int fred
[4]    ⍝: string msg
[5]     msg←'Hello from branches'
[6]     →(fred<6)↓L1
[7]    L1:→0/⍨fred<4
[8]     →(fred>2)↑Done
[9]     →0
[10]   Done:→Quit
[11]   Quit:msg,←' ... Quit reached'
      ∇
```

It has a couple of special comments, but otherwise looks just like the stuff we all had to write before James Wheeler smuggled control structures into APL+Win.

Let's convert this to C# and see what it looks like:

```
  tcc.ccr 'branches'

// ========== branches ==========
// Assorted branch styles

public string branches(int fred)
{
  string msg;  // Result
  msg = "Hello from branches";
  if (AE.EQ(0,(fred < 6))) goto L1;
 L1:if (fred < 4) return msg;
  if (fred > 2) goto Done;
  return msg;
 Done:goto Quit;
 Quit: msg = msg + " ... Quit reached";
```

Just for old-times' sake, I even exdent the labels one stop. Basically you can use simple branching instructions, but any attempts to do clever stuff that manipulates vectors of labels:

```
        cc.chk '→expr⍎lab'
         →      expr  ⍎      lab
   MARK  CTRL  NOUN  VERB  NOUN
  Unsupported computed branch AE.Rotate(expr,lab)
```

... are doomed to failure. Fortunately, these are usually very easy to redo.

Control structures R Us

Just how well do the APL structures map over into C#? This really is crucial to maintaining the readability of the generated code, and absolutely essential if we are ever to abandon APL altogether and maintain *RainPro* only in C#. Unlikely for the base graphics engine, but very likely indeed for things like the SVG output conversion. Let's start with the easy ones and build up from there:

```
     ∇ r←ifreturn x
[1]    ⍝ Simple Return example
[2]    ⍝:Public
[3]    ⍝: int x
[4]    ⍝: string r
[5]
[6]     :If x>0
[7]       r←'Out of here'
[8]       :Return
[9]     :Else
[10]      r←'Done'
[11]    :End
     ∇
```

```
// ============ ifreturn ============
// Simple Return example

public string ifreturn(int x)
{
  string r;  // Result
  if (x > 0) {
    r = "Out of here";
    return r;
  } else {
    r = "Done";
  }
  return r;
}
```

At this point, I will have upset about 50% of the C# programmers in the world. Just as we APLers have religious wars about ⎕IO, conventional programmers have wars about code layout conventions. I went for the old K&R style, mainly because it generates a lot less white space, and the lines match quite closely with the lines of APL code they came from. Here is a slightly more complex case:

```
     ∇ r←ifandifandifelse x
[1]    ⍝ Basic IF block example
[2]    ⍝:Public
[3]    ⍝: int r,x
[4]
[5]     :If x>0
[6]     :AndIf x>1
[7]     :AndIf x>2
[8]       r←0
[9]     :Else
[10]      r←12
[11]    :End
     ∇
```

```
public int ifandifandifelse(int x)
{
  int r;  // Result

  if (x > 0
    && (x > 1)
    && (x > 2)) {
    r = 0;
  } else {
    r = 12;
  }

  return r;
}
```

Again, the match is very accurate, and we get the same behaviour that the 'and' clauses are only evaluated if all the prior expressions succeed. Note in passing that the conversion process recognises where a C# native function (like *greaterthan* on two scalars) is adequate to do the job, and doesn't bother to call our APL

library for these. It is slightly depressing just how often this seems to happen in 'realistic' application code. Anyway, here's a *For* loop to see how that goes:

```
 r←forloop n;ct;stuff;nv
⍝ Basic looper
⍝:Public
⍝: int    n,stuff
⍝: string r

 nv←⍳32 ⋄ stuff←0

 :For ct :In ⍳⍴nv
   stuff+←nv[ct]+1000
 :End

 r←'Done ',⍕stuff
```

```
public string forloop(int n)
{
  string r;  // Result
  int   stuff;
  int[] nv;

  nv = AE.Range(32);
  stuff = 0;

  for (int ct = 1; ct <= nv.Length; ct++) {
    stuff = stuff + (nv[ct-1] + 1000);
  }
  r = "Done " + AE.Format(stuff);
  return r;
}
```

The conversion special-cases loops over *iota* as true *for(;;)* structures. Anything else turns into a C# *foreach()* block which is much more general, but less efficient.

A couple of other things to note here:

- we can generally forget monadic *rho* in favour of the *Length* property, which all arrays support automatically.
- indexing by scalars is fine, but C# is origin-0 so we must subtract 1, always.
- character vectors are mapped to C# strings, and the format primitive always returns a string, rather than a vector of characters. This allows us to map most string catenations to the C# + operator which adds hugely to readability, but does have some annoying side effects. Another one to upset the purists, but the pragmatists will understand.

So far, so good. *While* loops fall out easily enough:

```
 whileleave arg
⍝ Check out conversion
 :While true
   ⎕←arg
   :If arg<3
     :Leave
   :End
   arg←arg-1
 :EndWhile
```

```
public void whileleave(int arg)
{
  while (true) {
    AE.Display(arg);
    if (arg < 3) {
      break;
    }
    arg = arg - 1;
  }
}
```

Note in passing that we chose to treat *assignment to quad* as a function (which echos its argument to the console, then returns it untouched).

That leaves *Select/Case* as the only slight oddball.

```
 r←caselist item;opt
⍝ Test caselist block
⍝:Public
⍝: string r,item,opt

 r←'(not set)'
 opt←'One'

 :Select item
 :Case 'fat'
   r←'fatter'
 :CaseList 'cat' 'moggy'
   :Select opt
   :Case 'One'
     r←'Hello cat'
   :Case 'Two'
     r←'Hello moggy'
   :Else
     r←'not my cat'
   :EndSelect
 :CaseList 'bonzo' 'snapper'
   r←'not my dog'
 :Case 'sat'
   r←'Sitting'
 :End
```

```
public string caselist(string item)
{
  string r;  // Result
  string opt;
  r = "(not set)";
  opt = "One";
  switch (item) {
    case "fat":
      r = "fatter";
      break;
    case "cat": case "moggy":
      switch (opt) {
        case "One":
          r = "Hello cat";
          break;
        case "Two":
          r = "Hello moggy";
          break;
        default:
          r = "not my cat";
          break;
      }
      break;
    case "bonzo": case "snapper":
      r = "not my dog";
      break;
    case "sat":
      r = "Sitting";
      break;
  }
  return r;
}
```

This one (as you can see) was a touch tricky to get right. You have to add in all those *break;* statements, as well as chopping up the *CaseList* clause into a list of *cases*. However (major bonus) in C# we can use **strings** in a case block which is a huge step forward from Java where you can only *switch/case* on integers.

Of course, control structures are not the only block element supported by APL. There is *each* to cope with, and in Dyalog APL we might encounter *composition* and *dynamic functions*, probably used in combination with operators like *each* or *reduction*. How do we get around these?

Various Operators and Inline Dynamic Functions

In sketching the initial design for the C# generator, we had decided to follow A+ and simply 'can' all the common reductions, scans and inner/outer products as primitives. This remains a good approach (for example, expressions like `+\vec` can run much faster when special-cased), but most modern APLs make life a little

harder by permitting user-defined functions to the left of operators like reduction and scan, for example:

```
      plus/⍳3
  6
```

... where `plus` has the obvious (dyadic) definition. The way interpreters handle this is to create a temporary 'derived function' *plus-reduction* and then proceed to apply this monadically to the right argument. The only solution I can see is to copy this approach exactly, for example:

```
  cc.chk'plus/⍳3'
```

```
int plusReduce(int[] myrarg)  // Monadic Reduce
{
 int res = myrarg[0];
 int elem = myrarg.Length;
 if( elem > 1 ) {
   res = plus(myrarg[elem-2],myrarg[elem-1]);
   for (int i=elem-3;i>=0;i--)
     res = plus(myrarg[i],res);
 }
 return res;
}

 plusReduce(AE.Range(3));  int
```

The target function is type-checked, and assuming it is dyadic, and supports matching left and right argument types (both must be of rank one less than the argument to the generated temporary function) then we just make a block of C# code to do what reduction does (walk back down the array from right to left) and return as the parsed expression a simple monadic call to it.

Clearly, the same approach will work for *Scan*, but *Each* is a lot nastier, as the operator itself can be monadic or dyadic, which affects the acceptable valence of its function argument(s). Here is a simple example of the dyadic case:

```
  cc.chk '3 take ¨(1 2)(3 4 5)'
```

```
int[][] takeEach(int mylarg,int[][] myrarg)  // Dyadic Each
{
 int[][] res = new int[myrarg.Length][];
 for (int i=0;i<myrarg.Length;i++)
   res[i] = take(mylarg,myrarg[i]);
 return res;
}

 takeEach(3,new int[][]{new int[]{1,2},new int[]{3,4,5}});  int[][]
```

... and when we start allowing dynamic functions, you really have to cross your fingers and hope it works:

```
  cc.chk '3 {α↑ω} ¨(1 2)(3 4 5)'

int[] noname_d1(int alpha, int[] omega) { return AE.Take(alpha,omega); }  // Dfn dyad

int[][] noname_d1Each(int mylarg,int[][] myrarg)  // Dyadic Each
{
 int[][] res = new int[myrarg.Length][];
 for (int i=0;i<myrarg.Length;i++)
   res[i] = noname_d1(mylarg,myrarg[i]);
 return res;
}

 noname_d1Each(3,new int[][]{new int[]{1,2},new int[]{3,4,5}});  int[][]
```

The principle is the same, but we must make a unique name for the generated Dfn, as C# (unfortunately) does not permit local functions. In real code, the Dfn would very likely be local, and would be prefixed with the name of its caller.

Oh, I nearly forgot. Now we have dynamic functions, I just cheat with *composition* and pretend it was a dfn all along:

```
       cc.chk '3∘↑¨(1 2)(3 4 5)'
        3     ∘     ↑      ¨      →(1 2) (3 4 5)⊢
  MARK  NOUN  CONJ  VERB   ADVB   NOUN

  int[] noname_d1(int[] omega) { return AE.Take(3,omega); }
```

... and then everything carries on as before. Notice that I have followed the J rules here, and am treating *compose* as CONJ rather than ADVB – I think it probably is something 'special' in the syntax, rather than just another operator.

Multadic Functions

Every so often, you paint yourself well into a corner with a series of perfectly reasonable decisions, all of which seem fine at the time. Remembering that C# is strictly scoped, we have a cast-iron rule that a function cannot see local variables in its caller. Fine, we just need to be sure to pass in everything as arguments, which is good APL practice. The snag is that APL has only one way to do this, which is to construct a nested array from the argument set, pass this in as a single item, and then unpick it inside the called function.

"Oh bother", said Pooh. We just took the decision to support arrays of arrays which are homogeneous all the way down (C# types like int[][]) and to avoid the overhead of the general nested structure, which C# represents as object[] and which implies a lot of runtime expense in recovering (and casting) the contents.

When in doubt, cheat! Here is the dodge we use to get around this particular fix:

```
      ∇ msg←multadic arg;first;second;third
 [1]   ⍝ Sample multi-argument function
 [2]   ⍝:Args first,second,third
 [3]   ⍝: double first
 [4]   ⍝: double second
 [5]   ⍝: string third,msg
 [6]
 [7]    (first second third)←arg ⋄ first←1⌈first
 [8]
 [9]    msg←'Third argument was ',third
      ∇
```

... which is converted to C# as:

```
string multadic(double first,double second,string third)
{
  string msg;  // Result
  first = AE.Max(1,first);
  msg = "Third argument was " + third;
  return msg;
}
```

... and called appropriately:

```
      cc.chk 'multadic 12.3 23.4 ''Hello'' '
       multadic  ⊣12.3 23.4 'Hello'⊢
MARK  VERB      NOUN
multadic(12.3,23.4,"Hello");  string
```

... which is why the word-formation stage is very concerned to flag up strands. If the parser finds itself formatting a strand as part of a function argument, it wraps it in parentheses and separates the elements with commas. When a function with an *Args* comment is processed, the header is built from this comment, and the (hopefully obvious) strand assignment is eliminated from the generated code. Note the difference from:

```
      cc.chk 'qqq←12.3 23.4 ''Hello'' '
       qqq   ←     ⊣12.3 23.4 'Hello'⊢
MARK  NOUN  ASGN  NOUN
qqq = new object[] {12.3,23.4,"Hello"};  object[]
```

So, you can have nilads, monads, dyads and multiads in this strange new world. What you cannot have is a function with a complex left argument as well as a complex right argument. I am sure I could cope with these if I had to, but we have to draw the line somewhere.

Structures and Simple Classes

From this point, the way the code-conversion works is quite specific to Dyalog version 10. Everything else could run in Dyalog, APL+Win or APLX with very little modification. What we have discovered along the way is that there are quite a few places where the ability to instance namespaces (and pass references as function arguments) is extremely helpful in migrating towards a pure C# coding style. A good example is the *StringBuilder* class, which comes free with C# and is extremely useful when building up longish strings (like the SVG representation of a towerchart) efficiently. To use it, you simply create a new instance, run its *Append* method as often as you want, and eventually collect the result by running its *ToString* method.

Something very similar in APL could look like:

```
      ∇ str←sbt;fluff;nl;word
[1]    ⍝ Give the StringBuilder a whirl
[2]    ⍝:Public
[3]    ⍝: string str,word
[4]     nl←⎕AV[4]
[5]     fluff←new #.StringBuilder
[6]     fluff.Append'Hello, world?',nl
[7]     fluff.Append'Wow!',nl
[8]     fluff.Append'Farewell, cruel world.',nl
[9]    ⍝ Check string substitution
[10]    fluff.Replace'world' 'Universe'
[11]   ⍝ Return completed string
[12]    str←fluff.ToString
      ∇
```

Of course, we could use the Dyalog-10 capability of talking to .Net to get straight to the real thing, but somehow this feels like overkill. Why not just make a `StringBuilder` namespace with a couple of utterly trivial functions called `Append` and `ToString` that do the same job? Then we need a mildly hairy function called `new` to get a true instance of the namespace. This does some pretty horrid stuff with `⎕NS` to copy all the functions and variables, and then it runs the constructor, if present. The constructor is defined to be any function with the same name as the class. In C#, we have exactly this behaviour already available via the *new* keyword, so when I run the code-generator on this function:

```
public string sbt()
{
  string str;  // Result
  char         nl;
  string       word;
  StringBuilder fluff;
```

```
  nl = '\n';
  fluff = new StringBuilder();
  fluff.Append("Hello, world?" + nl);
  fluff.Append("Wow!" + nl);
  fluff.Append("Farewell, cruel world." + nl);
 // Check string substitution
  fluff.Replace("world","Universe");
 // Return completed string
  str = fluff.ToString();
  return str;
}
```

... it looks remarkably similar, and behaves in exactly the same way. Of course you can use the same trick to define little structures (the legend definition in *RainPro* is a good example) as namespaces, and include these in the generated DLL. It seems moderately efficient in the source APL, and certainly moves one some way towards an object-oriented approach. Definitely worth doing.

Putting it all Together - some Benchmarks

OK, you can convert a line of code, so you can convert a function, so you can convert a whole namespace full of functions, and call it a class. Alternatively you can just take a few functions at random, write these to your source file and call these a class for testing purposes. Here is the way I am doing this at the moment:

```
'd:\tools\aplc\bench' Make csΔbench    ⍝ The Eratos benchmark
Bench - done header ...
(#.Eratos)(#.Looper)
All Done.
1303 bytes written to d:\tools\aplc\bench.cs
```

... where `csΔbench` is mostly just raw C# stuff with a few 'special' lines:

```
using System;
using AE=Causeway.SharpArrays;

public class MainClass
{
  static public void Main()
  {
    Bench test = new Bench();
    Console.WriteLine("Off we go");
    Console.WriteLine((test.Eratos(24000)).Length);  // 1 sec
    Console.WriteLine(test.Looper(160000000));       // 1 sec
  }
}

!Bench=Eratos,Looper   ⍝ Benchmarks, assorted
```

The last line creates a C# class called *Bench* out of the functions listed after the '=' sign. Here is the complete C# source which it produces:

```
using System;
using AE=Causeway.SharpArrays;

public class MainClass
{
  static public void Main()
  {
    Bench test = new Bench();
    Console.WriteLine("Off we go");
    Console.WriteLine((test.Eratos(24000)).Length);  // approx 1 sec
    Console.WriteLine(test.Looper(160000000));       // approx 1 sec
  }
}

public class Bench
{
 // ============ Eratos ============
 // Returns primes up to n by the sieve of Eratosthenes

  public int[] Eratos(int max)
  {
    int[] primes;  // Result
    int   check;
    int[] candidates;

    primes = AE.Ravel(1);
    candidates = AE.Drop(1,AE.Range(max));
    while (0 < candidates.Length) {
      primes = AE.Join(primes,check = candidates[0]);
      candidates = AE.Compress(AE.LT(0,AE.Residue(check,candidates)),candidates);
    }
    return primes;
  }

 // ============ Looper ============
 // Simple looper to test speed when we do very little!

  public string Looper(int n)
  {
    string r;  // Result
    int ct;
    ct = 0;
    while (ct < n) {
      ct = ct + 1;
    }
    r = "Done looping";
    return r;
  }

}  // End of class Bench
```

The *Looper* example copies exactly the 'daft benchmark' which the STSC guys used back in 1982 to put an upper bound on the possible speedup of APL code. They got a factor of 350 (compiling to 370 Assembler) which I was hoping to get close

to. In fact we do rather better – to get this to run for 1sec in C# I have to loop 160,000,000 times and (yes, I did try it) this takes a little over 17min in Dyalog APL, which gives me a factor of around 950. The *Eratos* example is very comparable with the Dyalog version, so your 'typical' commercial APL will be somewhere in between. I think we are getting about ×12 on the chart viewers (which basically convert PostScript macros into XML strings) and maybe ×3 on the graph-generation (the ch namespace) which is playing more to APL's strengths. A simple simulation (like the *Monty Hall* problem in the next *Vector*) would probably get you a factor of around 100 as it is mostly interpreter overhead with very little exploitation of the array stuff.

Wrap Up

Given a lightweight library (currently 504K, and unlikely to grow much beyond that) of APL-like functions, converting an APL namespace to a C# class turns out to be rather easier than expected. So far, our experience has been that the older (and hence simpler) the APL code, the easier the job becomes.

A major exception to this rule would be an application written in pure Dfns, which could be handled very cleanly, and rather more completely than an application written in traditional APL style. For example, D has just one control-structure (the guard) and one method of error-trapping (the error-guard) both of which convert very simply to C#. D has strictly lexical scope, which is much closer to the C# model, and D programmers take a pride in avoiding re-use of local variables. Ambivalent functions are easy to spot syntactically, and should be easy to create overloads for. Of course a D compiler should be written in D, and then (naturally) used to compile itself. Maybe someone with time on their hands would like to give it a try.

Background Reading

[1] Kenneth E. Iverson, *Formalism in Programming Languages*, in *A Sourcebook of APL*, APL Press 1981

[2] Clark Wiedmann, *Steps Toward an APL Compiler*, APL79 Conference Proceedings (APL Quote Quad Vol.9 No.4 June 1979)

[3] Philip R Chastney, *The Hunting of the Snark*, APL82 Conference Proceedings (APL QQ Vol.11 No.1 Sept 1982).

[4] Clark Wiedmann, *A Performance Comparison between an APL Interpreter and Compiler*, APL83 Conference Proceedings (APL QQ Vol.13 No.3 March 1983).

[5] Timothy A Budd, *An APL Compiler for the UNIX Timesharing System*, APL83 Conference Proceedings (APL QQ Vol.13 No.3 March 1983).

[6] J Philip Benkard, *Valence and Precedence in APL Extensions*, APL83 Conference Proceedings (APL QQ Vol.13 No.3 March 1983).

Appendix: Current Restrictions and Known Issues

Inlined dfns must not span multiple lines of the calling function.

Arrays are simple and homogeneous or genuinely nested, not heterogeneous.

Primitives are limited to depth 2 (we only allow string[][] and int[][] and double[][]). Deeper arrays are allowed, but must be handled with each or loops. The only primitives supported on arbitrary arrays are Pick, Index, Reshape at present. Probably some others like Compress, Drop, Mix, Split will be added as required. Scalar primitives do not pervade deep arrays. Probably you will always need 'each' here.

You need to be careful about rank. Singletons are not the same as scalars! Use `⍴pmat` not `1⍴pmat` in general!

Text constants are treated as strings, unless they only have one character. If you want to get 'inside' a string, you must explode it into a char[] which is done with ravel (`,`)

```
mycharvec←,'Hello, world'
```

To turn it back into a string, you must format it! In general, strings are treated as scalar 'atoms' so `3⍴'Hello'` is an array of 3 strings "Hello" "Hello" "Hello" and so on. The only exception is indexing, which looks inside and returns the selected characters as a char[] rather than as a string. This adds a few annoyances, but generates much nicer looking C# code which probably runs faster too.

Indexed assignment into arrays can cause strange side-effects. Arrays are passed by reference in normal C# fashion, so calling function `foo` and passing `myarray` ...

```
foo myarray
myarray[2] ← 12
```

will overwrite the 2nd element of `myarray` IN THE CALLING FUNCTION. Be careful!! If you really want to do this, make sure to clone the array first, for example

```
newarray ← array[]
```

Then mess with your new array. Similarly ...

```
vec2 ← vec1
vec1[1] +← 1
```

will increment `vec2[1]` as `vec2` is really only a pointer. Use `vec2 ← vec1[]` to be sure you make a copy.

C# keywords (like while, new, int, static) should be avoided in variable/function names.

The only Goto forms are:

Goto Lab or GoTo 0
Goto(expression){compress | take | drop}Lab
Goto Lab <slash><commute> expr

No execute (but you can use ⎕FI to get numbers from text chars)

No ⎕ELX or ⎕ALX

Only simple `:Trap 0 ... :else ... :EndTrap` is supported.

Dyalog ⎕SIGNAL ignores the number, so behaves like ⎕ERROR in APL+Win

No ⎕FMT (use ⍙PF or write your own, or provide C# alternative expression)

No ⎕FX or ⎕DEF

No ⎕VFI - use ⍙FI and ⍙VI in Dyalog APL to mimic ⎕VI and ⎕FI

No dyadic thorn. Use ⍙PF to pass a C# 'picture' on the left instead.

Residue - implemented with the C# definition so is the same as APL for +ve arguments but will give different results for -ve arguments. Be careful!

For now, all system variables are read-only:

⎕TCNL	Newline
⎕AV[4]	ditto
⎕TCLF	Linefeed
⎕IO	1 always
⎕ML	3 always (matching APL2 and APL+Win)
⎕PP	PrintPrecision - machine default
⎕PW	PrintWidth - irrelevant
⎕CT	ComparisonTolerance is machine precision, always
⎕RL	RandomLink is meaningless (real random numbers only)
⎕AV	AtomicVector - not supported yet - should be avoided

so ⎕IO is fixed at 1 here. It could easily be 0 but will never be switchable.

Ambivalent functions are not supported, as ⎕NC makes no sense here.

No ⎕SHADOW (localisation must be determined at the function level)

No user-defined operators.

Arrays are limited to 0 1 2 dimensions (so we do scalars, vectors, matrices) only. Dyadic transpose is therefore not supported at all.

Reshape takes a SCALAR left arg to create a vector result. Passing a length-1 vector will always be an error, as it will attempt to make a matrix of rank-1. For symmetry the shape of a vector is a scalar. You cannot take the shape of a scalar. Rank = ⍴⍴ is special-cased as a primitive, and works as you would expect.

Take and Drop only have singleton left arguments for now. To work with matrices you must use `2↑[1]mat` here. Accidental enhancement is `2↑mat` which works on the first axis by default in true J,A+ style.

There are limited options to the left of assignment viz

```
⎕←stuff                 ⍝ The only form of console output
var←stuff
arr[this;that]←stuff
arr[this;]←stuff
(msk/arr)←stuff
((expr)⊃arr)←stuff
(one two three)←stuff   ⍝ Parentheses are required here
```

Strand assignment *is* allowed – but you can declare a multi-argument function which eliminates most of these. The resulting strand (usually line-1) is simply eliminated from the generated code. Strands of names and constants seem OK, but avoid if possible as there may be all sorts of odd edge effects when tokenising these. Session echo must be explicit with assignment to ⎕, even at the left-hand end of a line.

If and *AndIf* must be on adjacent lines. No support for

```
:If summat
   do this
:AndIf summatelse
   do this as well
:Else
   do that
:End
```

which works in Dyalog (but not +Win, I think). No support for double-testing:

```
:While (cond1)
  stuff
:Until (cond2)
```

all other structures are OK, but note that Select/Case is limited to integers and strings and the *case* clauses must all be constant expressions.

Elliptic Curves and Factoring II

by Cliff Reiter (reiterc@lafayette.edu)

We previously noted [5] that factoring integers has become a topic of intense interest because some modern security schemes depend upon the assumption that factoring the product of two sufficiently large primes is implausible [1,3,6].

In that note we developed some J facilities for finding factors. The most sophisticated technique utilized elliptic curves for factorization. We did not combine those techniques into a single function that factored integers since the problem of how to identify primes as such was not resolved in that note. We will consider a fast probabilistic method for identifying primes in this note then combine those with the facilities from the previous note to create a function that routinely gives the prime factorization for 25 digit numbers.

Pseudoprimes and Carmichael Numbers

A classic theorem called Fermat's Little Theorem (FLT) states that for a prime number, p, and a base a not divisible by p, we get $a^{p-1} \equiv 1 \bmod p$. For example, since 101 is prime, the following residues must be 1.

```
   101 | 2^100x
1

   101 | 3^100x
1
```

Since J can compute these modulo powers very efficiently if it recognizes them, we will take care to use tacit composition of the residue with the exponential.

```
   3 (101&|@^) 101x-1
1

   fltq=: 4 : '1 = x. y.&|@^ y.-1'

   2 3 4 fltq 101
1 1 1
```

FLT only holds when the base is relatively prime to the number being tested. Thus, the following does not show that 101 is composite (it is prime).

```
    101 fltq 101
0
```

Thus, before applying FLT one should check that the base is relatively prime to the number (if the gcd is a nontrivial factor of the number, then the number is, of course, composite).

The function `fltq` tests whether the conclusion of FLT holds for the bases given as left argument for the number given as the right argument. Many composites can be recognized by the fact that FLT fails for some base. For example, the following makes it clear (3 times) that 143 is composite, even though no factors are given.

```
    2 3 4 fltq 143
0 0 0
```

However, 341 is also composite, but it looks prime when bases 2 and 4 are used but is shown to be composite by base 3.

```
    2 3 4 fltq 341
1 0 1
```

When n is composite such that $b^{n-1} \equiv 1 \bmod n$ we say that n is a pseudoprime to the base b.

It is possible that n is a pseudoprime to the base b for every base that is relatively prime to n. Such numbers are called Carmichael numbers. There is no hope that FLT can be used to recognize n as composite unless a base with a common factor is chosen. Thus, FLT is no better than actually having found a factor by trial division or whatever mechanism chose the trial bases.

One Carmichael number is 6601.

```
    2 3 4 5 fltq 6601
1 1 1 1
```

While Carmichael numbers are rare, there are infinitely many of them and 105,212 of them below 10^{12} [1].

MSR Test

While the FLT test fails to identify 6601 as composite unless a lucky base is chosen, there is a stronger version of the test that shows 6601 to be composite. It is known that for a prime p, the quadratic equation $x^2 \equiv 1 \bmod p$ has only $x \equiv 1$ and $x \equiv -1$ as

solutions. Thus, for a proposed base a, we compute $a^{\frac{n-1}{2}}$ and expect to get either 1 or -1 if n is prime. If n is prime we must. If not, we know n was composite. If we obtained a 1, and $\frac{n-1}{2}$ is even, we can consider $a^{\frac{n-1}{4}}$. If n is prime, we must get ±1. We see that 6601 fails this test with base 2.

```
   2 (6601&|@^) 6600x
1

      2 (6601&|@^) 6600x%2
1

      2 (6601&|@^) 6600x%2^2x
4509
```

In general, this process can be repeated until either a -1 is encountered, or there are no further factors of 2 to be removed from n-1. Re-organizing the arithmetic slightly, we get the following test.

MSR Probabilistic Primality Test

Suppose we factor $n = 2^k m$ where m is odd, then we compute the following:

$x \equiv b^m \bmod n$

If x is not congruent to ±1 mod n then use squaring to compute

$x \equiv b^{2^j m} \bmod n$ for $j = 1,2,\ldots,k-1$ or until $x \equiv -1 \bmod n$.

If any x computed is congruent to ±1 mod n, then we say that n passes Miller's test to the base b. Otherwise, it fails the test to the base b.

Below we implement the MSR method and see that even though 6601 is a Carmichael number, it fails the MSR test bases 2, 3, and 4.

```
   pwr2in=:3 : 0
j=.1
while. 0=(2x ^j)|y. do.
  j=.j+1
end.
j-1
)
```

```
   msr=: 4 : 0"0"1 0
b=.x.
n=.x: y.
k=.pwr2in n-1
m=.(n-1)%2x^k
x=.n&|@(b&^)m
if. 1 = x do. 1 else.
j=.0
while. (x~:n-1)*.j<k do.
  x=.n|x^2
  j=.j+1
end.
if. x = n-1 do. 1 else. 0 end.
end.
)

   2 3 4 msr 6601x
0 0 0
```

Primes pass the MSR test to every base not divisible by the prime. It is known [1] that the likelihood that a composite number passes the MSR test base b is less than $\frac{1}{4}$. Thus, if it passes the MSR test to s bases, we consider it prime with the high level of confidence. The likelihood of an error is less than $\left(\frac{1}{4}\right)^s$. It is conjectured that all odd composite integers have a base $b < 2\ln^2(n)$ for which the Miller test fails. We will consider an integer prime if it passes Miller's test for bases consisting of the first 100 primes. If higher reliability is desired, it is easy to raise the number of bases to $2\ln^2(n)$ or higher.

Moreover, there are other probabilistic tests for primality that are of complexity comparable to Miller's test and that give further reassurance of the primality of n. See the discussion of Fibonacci and Lucas pseudoprimes in [1].

The following gives and illustrates our probablistic primality test.

```
   primeq=: 3 : 0"0
100 primeq y.
:
b=.p:i.x.>.3
if. y. e. b do. 1 return. end.
if. -.*/(2{.b) msr y. do. 0 return. end.
*/(2}.b) msr y.
)
```

```
    primeq 2 4 101 6601
1 0 1 0
```

Now we turn to the function that puts together the factorization techniques from [5].

Factoring

We have seen factorization techniques that are effective at finding factors of numbers with up to around 25 digits reasonably quickly. The techniques discussed in that note included the following.

- Trial division, which we used to find small prime factors; typically including primes up to 100.
- Pollard p-1, often good for quickly finding small factors; we used it to find easy factors with a few digits.
- Pollard rho, also often good for finding small factors; takes somewhat longer and may find somewhat larger factors.
- Elliptic curve factorization; good for larger factorizations but may take minutes as opposed to seconds for factors with a dozen digits.

Here we factor integers by first removing small factors, using trial division. The result is organized into two lists: one of prime factors and the other of unfactored terms. We use `primeq` to identify factors that are prime and which should be moved to the list of prime factors. While the list of unfactored composite terms is non-empty, we try to factor its first element. We apply Pollard p-1, then Pollard rho and then the elliptic curve method.

```
   factor=: 3 : 0"0
'ps fs'=.fac_smpr y.
if. fs=1 do. fs=.i.0 end.
q=.primeq fs
ps=.ps,q#fs
fs=.(-.q)#fs
while. 0<#fs do.
  f0=.{.fs
  'g v'=.fac_p_1 f0
  if. g e. 1,f0 do.
    'g v'=.fac_rho f0
    while. g e. 1,f0 do.
      'g v'=.2{.>{.fac_ecmj f0
      end.
    end.
  fs=.(}.fs),g,f0%g
  q=.primeq fs
  ps=.ps,q#fs
  fs=.(-.q)#fs
  end.
/:~ps
)
```

We illustrate this function by factoring a random integer.

```
   ]x=:randi 25
92138616334158595194152 44

   factor x
2 2 307 26821 17977907 15560703359
```

A script containing the definitions of all these functions is available at [4].

The function `factor` is reasonably efficient. When we factored a thousand random 25 digit integers we found the median time was around 0.5 seconds while the average was near 8 seconds. Only a few factorizations were "hard"; the hardest factorization took 540 seconds. Timings were done on a 750 MHz PC using J5.01b [2].

If we take the ceiling of base 2 logarithm of the time in seconds to factor, we get the following distribution.

```
   $x=:randi 1000#25

   fac_time=:3 : 0"0
t=.time 'z=:factor y.'
t;z
)
```

```
   $zz=:fac_time x
1000 2

   |:({.,#)/.~>.2^./:~>{."1 zz
_2  _1   0  1  2  3  4  5  6  7  8 9 10
31 384 283 94 76 35 15 33 16 18 10 3  2
```

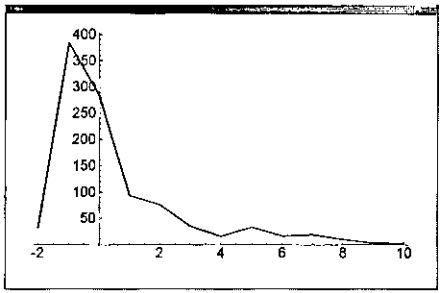


Figure 1. Number of times $\log_2(t)$ seconds were used to factor 25-digit random integers among 1000 trials.

We see that a majority of the factorizations required less than one second. We now look at a "hard", 27-digit, factorization problem and run it 100 times. We get the following distribution, again taking the ceiling of the base 2 logarithm of the time in seconds.

```
   mm=:152415787533657061564561727x

   factor mm
1234567890133 123456789012419

   $ww=:fac_time 100#mm
100 2

   |:({.,#)/.~>.2^./:~>{."1 ww
5  6  7  8  9 10 11 12
6 10 17 16 14 18 15  4
```

Figure 2 shows that distribution. It turns out that the fastest factorization time was 20 seconds and the longest was 3000 seconds. The median was 286 seconds and the average was 540 seconds.

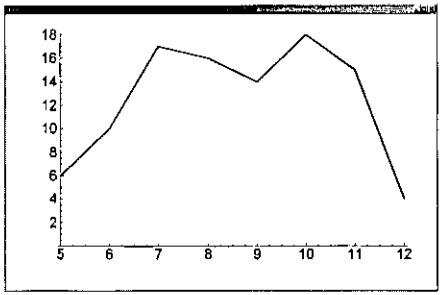


Figure 2. Number of times $\log_2(t)$ seconds were taken to factor a "hard" 27-digit integer from 100 trials.

Conclusions

We see that we can efficiently decide the primality of large integers with high reliability. Techniques for finding nontrivial factors include Pollard p-1, Pollard rho, and elliptic curve factorization. Implementing and combining those techniques in J leads to a factorization function that is quite effective on 25 digit numbers and is often effective for larger numbers.

References

[1] R. Crandall and C. Pomerance, *Prime Numbers: A Computational Perspective*, Springer, New York, 2000.

[2] Jsoftware, J501b, *http://www.jsoftware.com*.

[3] C. Reiter, With J: Public Key Crytography, *APL Quote Quad*, 32 2 (2001) 21-24.

[4] C. A. Reiter, Elliptic curve factoring script, *factor_ecj.ijs*, *http://www.lafayette.edu/j/vector/index.html*.

[5] C. Reiter, With J: Elliptic Curves and Factoring I, *APL Quote Quad*, submitted.

[6] R. L. Rivest, A. Shamir, and L. Adlemen, A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM*, 21 2 (1978) 120-126.

Index to Advertisers

All queries regarding advertising in VECTOR should be made to Gill Smith, at 01439-788385, Email: apl385@compuserve.com.

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 (hardcopy with diskette as appropriate) to the Vector Working Group via:

Vector Administration, c/o Gill Smith
Brook House
Gilling East
YORK, YO62 4JJ
Tel: +44 (0) 1439-788385
Email: apl385@compuserve.com

Authors wishing to use Word for Windows should contact Vector Production for a copy of the APL2741 TrueType font, and a suitable Winword template. These may also be downloaded from the Vector web site at www.vector.org.uk

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 YO62 4JJ. Please also copy us with all electronically submitted material so that we have early warning of possible problems.

Subscribing to Vector

Your Vector subscription includes membership of the British APL Association, which is open to anyone interested in APL or related languages. The membership year runs from 1st May to 30th April. 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: _____

UK private membership	£20	❑
Overseas private membership	£22	❑
Airmail supplement (not needed for Europe)	£4	❑
UK Corporate membership	£100	❑
Corporate membership overseas	£110	❑
Sustaining membership	£500	❑
Non-voting UK member (student/OAP/unemployed)	£10	❑

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

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:
BAA, c/o Rowena Small, 12 Cambridge Road, Waterbeach, CAMBRIDGE CB5 9NJ, UK
Fax: +44 (0) 1653 697719

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.

2004/2005 Committee

Chairman	Adrian Smith 01439-788385 apl385@compuserve.com	Brook House Gilling East YORK YO62 4JJ
Secretary	Anthony Camacho 0117-9730036 acam@blueyonder.co.uk	11 Auburn Road Redland BRISTOL, BS6 6LS
Treasurer	Nicholas Small 01223-570850 treas.apl@bcs.org.uk	12 Cambridge Road, Waterbeach, Cambridge CB5 9NJ
Journal Editor	Stefano Lanzavecchia stf@apl.it	c/o APL Italiana Corso Vercelli 54 20145 – Milano, Italy
Education	Dr Alan Mayer 01792-295779 a.d.mayer@swansea.ac.uk	European Business Management School, Swansea University, Singleton Park, SWANSEA SA2 8PP
Webmaster	Bob Hoekstra 01483-771028 bob.hoekstra@hoekstrasystems.ltd.uk	Dominique, Salisbury Road WOKING, Surrey GU33 7UR
Activities	Ray Cannon 01252-874697 ray_cannon@compuserve.com	21 Woodbridge Rd, Blackwater Camberley, Surrey GU17 0BS
Projects	Stephen Taylor sjt@lambenttechnology.com	
Administration	Rowena Small 01223-570850 treas.apl@bcs.org.uk	12 Cambridge Road, Waterbeach, Cambridge CB5 9NJ

Journal Working Group

Editor:	Stefano Lanzavecchia	see above
Production:	Adrian & Gill Smith	Brook House, Gilling East, YORK (01439-788385)
Advertising:	Gill Smith	Brook House, Gilling East, YORK (01439-788385)

Support Team: Anthony & Sylvia Camacho (0117-9730036),
Ray Cannon (01252-874697), Stephen Taylor (077-1340 0852)
Bob Hoekstra (01483-771028), Marc Griffiths

Typeset by APL-385 with MS Word for Windows
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
The Maltings, Castlegate
MALTON, North Yorks YO17 7DP
Tel: [illegible]
Fax: [illegible]
Email: sales@causeway.co.uk
Web: www.causeway.co.uk

Compass Ltd
Compass House
60 Priestley Road
GUILDFORD, Surrey GU2 5YU
Tel: [illegible]

Dyalog Ltd
[illegible] Close
[illegible] BASINGSTOKE
Hants RG[illegible]
Tel: [illegible]
Fax: [illegible]
Email: sales@dyalog.com
Web: www.dyalog.com

APL2000 Inc
Ops Research Group
Suite [illegible]
Rockville MD [illegible]
USA
Tel: [illegible]
Email: sales@apl2000.com
Web: www.apl2000.com

Insight Systems ApS
Nordre Strandvej 119G
DK-3150 Hellebaek
Denmark
Tel: [illegible]
Fax: [illegible]
Email: info@insight.dk
Web: www.insight.dk

RMW Computing
Hamilton House
1 Temple Avenue
LONDON EC4Y 0HA
Tel: [illegible]
Email: RMW@axis.com

MicroAPL Ltd
The Roller Mill, Mill Lane
Uckfield
E Sussex TN22 5AA
Tel: [illegible] Fax: [illegible]
Email: MicroAPL@microapl.demon.co.uk
Web: www.microapl.co.uk

Soliton Associates Ltd
Groot Blankenberg [illegible]
1082 AC Amsterdam
Netherlands
Tel: [illegible]
Fax: [illegible]
Email: sales@soliton.com

Kx Systems
555 Bryant St, Nr. 375
Palo Alto CA 94301 USA
Tel: [illegible]
Email: general enquiries: info@kx.com

In Europe: Simon Garland, SMP Technology
Email: simon@kx.com