VECTOR

100+ pages of the best in APL ...

The Journal of the British APL Association

ISSN 0955-1433
www.vector.org.uk
Vol.17 No.1 July 2000

A Specialist Group of the British Computer Society

Contributions

All contributions to VECTOR may be sent to the Journal Editor at the address on the inside back cover. Letters and articles are welcome on any topic of interest to the APL community. These do not need to be limited to APL themes, nor must they be supportive of the language. Articles should be accompanied by as much visual material as possible (b/w or colour prints welcome). Unless otherwise specified, each item will be considered for publication as a personal statement by the author. The Editor accepts no responsibility for the contents of sustaining members' news, or advertising.

Please supply as much material as possible in machine-readable form, ideally as a simple ASCII text file on an IBM PC compatible diskette or via email. APL code can be accepted in workspaces from I-APL, APL+Win, IBM APL2/PC or Dyalog APL/W, or in documents from Windows Write (use the APL2741 TrueType font, available free from Vector Production), and MS Word (any version).

Membership Rates 2000–2001

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

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

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

Sustaining membership is offered to companies trading in APL products; this is seen as a method of promoting the growth of APL interest and activity. As well as receiving public acknowledgement for their sponsorship, sustaining members receive bulk copies of VECTOR, and are offered news listings in each issue.

Advertising

Advertisements in VECTOR should be submitted in typeset camera-ready format (A4 or A5) with a 20mm blank border after reduction. Illustrations should be photographs (b/w or colour prints) or line drawings. Rates (excl VAT) are £250 per full page, £125 for half-page or less (there is a £75 surcharge per page if spot colour is required).

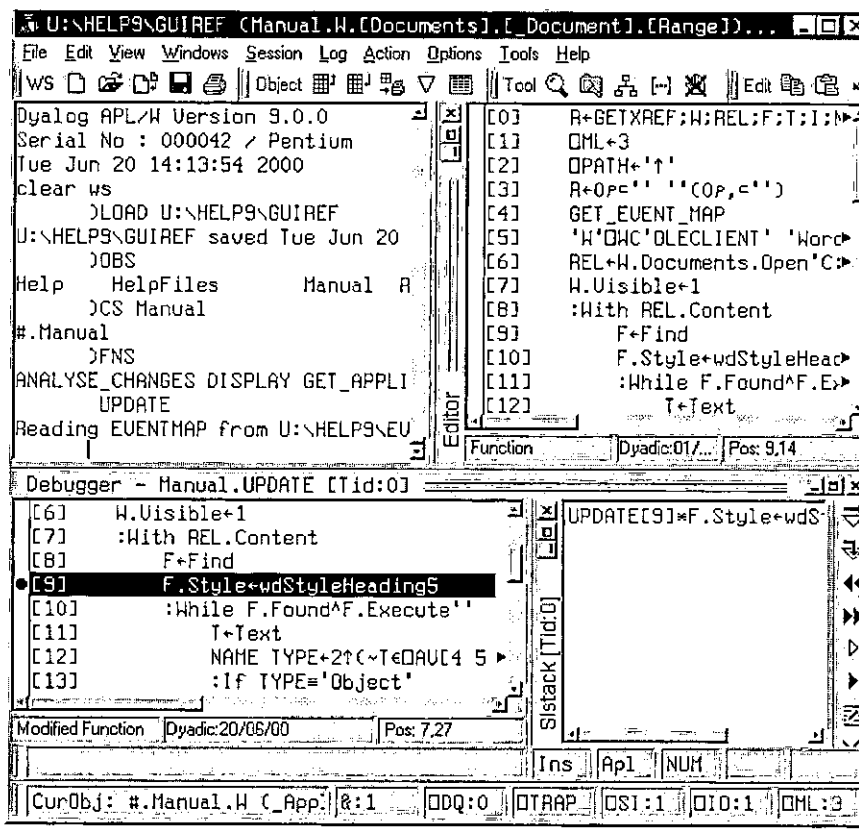
Deadlines for bookings and copy are given under the Quick Reference Diary. Advertisements should be booked with, and sent to Gill Smith, Vector Production, Brook House, Gilling East, YORK YO62 4JJ. Tel: 01439-788385.

Email: apl385@compuserve.com.

Contents

dyalog APL

The Definitive APL for Windows ™



Version 9 - an even better IDE(A)

http://www.dyadic.com

Dyadic Systems Limited, Riverside View, Basing Road, Old Basing, Basingstoke,
Hants. RG24 7AL, United Kingdom.
Tel:+44 1256 811125 Fax:+44 1256 811130 Email: sales@dyadic.com

Editorial

by Stefano Lanzavecchia

"A man is born gentle and weak.
At his death he is brittle and stiff.
Green plants are filled with sap.
At their death they are withered and dry.
Therefore the stiff and unbending is the disciple of death.
The gentle and yielding is the disciple of life.
Thus an army without flexibility never wins a battle.
A tree that is unbending is easily broken."
Tao Te Ching – Passage Number 76

Two years as the editor of Vector: it is hard to resist the temptation to collect a few thoughts in form of a balance. The joy and the excitation of working side by side with the group of nice and competent people composing the Vector Working Group is mitigated by the apparent lack of participation of our (my?) readers. I remember, back in the days when I was a young reader, I always enjoyed reading the letters and it was particularly intriguing when one thread would span several issues, involving several people, whose names I learned to recognise and respect. The number of letters sent to Vector has been dropping steadily and now is very close to a sad zero on average per issue. This is hard for me to understand: I am even ready to assume that the articles are uninteresting. But the reaction I would expect would be to be flooded by messages from angry subscribers asking for content more appealing. This dead calm is unsettling.

You certainly noticed that thanks to a French contributor, we even tried to revive the wonderful column of programming challenges, proposing the puzzle of the tessellation of the cube (see issue 16.3 page 44). After all these months the Working Group has received only three solutions, namely: mine, Adrian Smith's, and (hooray!) one coming from a reader. Once again, I cannot believe the challenge was so complex to distract too much all the busy professionals, lost in the aftermath of the Y2K doom, or so easy to be tagged as boring.

The funny thing is that it has never been so easy to communicate: in addition to the old ways (address book, paper and pen, envelope and stamp) we have at our disposal lists of addresses up-to-date online on the web pages, e-mail, SMS, when Fax and telephone are not enough. We are very laid back, so there's no need for formal introductions and elaborate greetings. We understand English, French, Italian, Danish and would not be scared by Spanish, Swedish, Norwegian, Finnish, German.

We have fun with what we do, but it would be much better if you had fun with us.

So, why don't you let us know what you think?

Quick Reference Diary

Date	Venue	Event
24-27 July 2000	Berlin	APL Berlin 2000

APL Berlin 2000

APL Berlin 2000 is an Array Programming Languages Conference, to be held in the Department of Computer Science at the Technical University of Berlin.

> The Conference Chairmen are Dieter Lattermann (dieter_lattermann@compuserve.com) and Conrad Hoesle-Kienzlen (conrad_hoesle@csi.com).

The Conference has a website at:

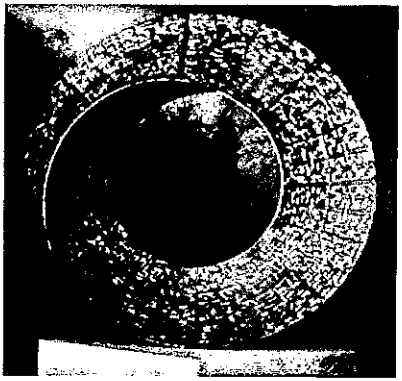
> http://stat.cs.tu-berlin.de/APL-Berlin-2000

mirrored at:

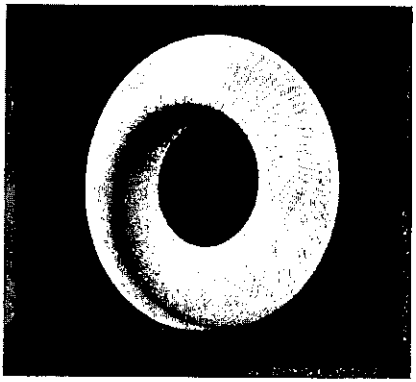
> http://www.lingo.com/APL-Berlin-2000

Dates for Future Issues of VECTOR

	Vol.17 No.2	Vol.17 No.3	Vol.17 No.4
Copy date	8th September	8th December	9th March
Ad booking	15th September	15th December	16th March
Ad Copy	22nd September	22nd December	23rd March
Distribution	October 2000	January 2001	April 2001



Umbilic Torus NC by
Helaman Ferguson, 1986

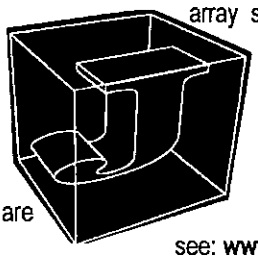


Umbilic Torus created in Jsoftware

For those who see the elegance in an algorithm, Jsoftware offers an OLAP programming language ideal for solving complex problems.

Jsoftware is the modeling tool of choice in the art of programming.

Jsoftware is an OLAP language. It's high performance interactive nature allows a programmer to concisely express algorithms when analyzing complex data sets.



The core language is based on a small set of simple yet consistent rules with many powerful facilities to define new operations. Built-in functions are highly optimized for operating on whole datasets at a time, allowing you to manipulate data far more easily than with conventional software. Also,sparse array support provides an efficient form of storage for very large data sets.

Make sure to join the JForum for a lively discussion on a variety of interesting J topics, see: **www.jsoftware.com/forum.htm**

Strand Software, Inc.
19235 Covington Ct.
Shorewood,MN 55331
(612) 470-7345
Fax (612) 470-9202
info@jsoftware.com

www.jsoftware.com

British APL Association News

Minutes of the Meeting of the Committee of the British APL Association held on 19 May 2000 at the Royal Statistical Society

It was agreed that the order of proceedings for the AGM would be:

1 Apologies
2 Minutes of the last AGM (printed in Vector)
3 Chairman's report
4 Treasurer's report
5 Questions to officers

It was agreed that we would not change the membership fees, though it was noted that the cost of sending Vector to the European members had risen significantly.

It was agreed that we would make 1000 or more copies of the proceedings CD which we have agreed to produce for APL Berlin 2000 and send a copy with each Vector and offer copies to other APL Associations for them to give to their members. This necessitates an increase in the budget to about £2500. The increase was agreed. It was suggested that we include on the CD the Vector archive, SAX for Linux, APL*PLUS/SE, TryAPL2, K-lite, the APL Statistics library, SigAPL free software, the Canadian utilities library, the FinnAPL idiom library, and maybe more.

The Chairman proposed that as it looked as if SigAPL/ACM might not sponsor APL Berlin 2000 the Chairman for 2000/2001 should offer further sponsorship if it was needed up to £1500 at risk. This was agreed nem con.

The committee agreed to co-opt Dave Phillips.

The Committee agreed that Anthony Camacho should represent it at the BCS technical board meetings.

The Committee re-appointed John Sullivan to audit the accounts for 2000-2001.

Anthony Camacho 20th May 2000

Minutes of the AGM of the British APL Association held at the Royal Statistical Society from 2pm to 2.15 pm on 19 May 2000

Apologies were received from Ian Clark and Stefano Lanzavecchia.

The minutes of the last AGM as printed in Vector Vol.16 No.1 were approved.

The Chairman reported as follows:

British APL Association Chairman's report on the year 1999-2000

The APL Association survives but is still slowly shrinking. We have about twenty members less than last year. There are some things we can congratulate ourselves about and I prefer to recall these.

The jewel in our crown is Vector, widely acknowledged to be the best APL magazine, worldwide. Vector has appeared regularly and the working group is confident that we can keep up the work for some while to come. Of course we are always looking for more and better articles to print and we encourage everyone to contribute. To those members (and it is not an empty set) who wish us to print more APL and APL2 we have to say that the magazine is bound to reflect the views of its authors at least as much as those of its readers and that it will continue to do so until the Editor is in the happy position of being able to pick the best and most suitable half of the articles submitted. Only our members could put him in that position, so it is up to you.

My thanks to Stefano Lanzavecchia and the Vector working group for an excellent year's work.

In my opinion, a major reason for the success of Vector is the working group. It is a large and friendly group of people that meets four times a year. The size of the group means that the task of obtaining articles is spread among many (a sole editor might find asking continually for articles becomes tedious) and a large group has a wider ring of acquaintances that can be asked. The friendliness of the group makes the quarterly meetings a pleasure to attend – and the Association does its best to help by paying for appropriate refreshments. It is an open group and we would be glad to welcome anyone interested in seeing what it is like to come and see. If you do, you won't be bullied into doing things that you don't want to.

Since the Vendor Forum following last year's AGM, we have not held a meeting. The main problem is finding a time, place and subject that will attract a large enough audience to justify the effort our speakers would have to devote to

producing good presentations. If there is a subject on which you would like to arrange a meeting, please contact Jon Sandles and he will help you, if possible, to realise your plan.

Your committee for next year is as follows:

Chairman	Adrian Smith
Secretary	Anthony Camacho
Treasurer	Nicholas Small
Editor	Stefano Lanzavecchia
Activities	Jon Sandles
Webmaster	Ray Cannon
Education	Ian Clark
Projects and Publicity	Alan Mayer

I thank last year's committee for their efforts, in particular Ajay Askoolum, who is retiring as Secretary, for his three years work in that post. Unfortunately I am now going to have to do the job myself, at least until we can find a volunteer! I am also grateful for the efforts of Rowena Small and Gill Smith who are paid to look after the membership and Vector administration.

This year the APL conference is in Berlin. Your association has agreed to sponsor the proceedings: we will produce a CD and loose-leaf sheets of paper for the delegates. As the CD has enough capacity for a considerable amount of further material, we are hoping to include the Vector archive (to the extent that it is available in electronic form), one or more free APL interpreters and as much as possible of the additional and tutorial material from the conference. We are hoping to send everyone a copy of the CD with the Vector containing the conference report.

Financially, we continue more than solvent. We have assets of over £40,000. For some years we have continued to keep the membership rates steady although it has meant that our expenditure exceeds our income. Whereas this obviously cannot continue for ever, your committee has felt that so long as we have sufficient funds to cover all foreseeable needs, we might as well keep our subscriptions low and encourage people to join.

Anthony Camacho 18 May 2000

Treasurer's report

Nicholas Small reported: he circulated a short set of accounts:

British APL Association - summary of annual accounts 1999/2000

Summary of income and expenditure/receipts and payments:

	1999/2000 (R&P) £	1999/2000 (I&E) £	1998/99 (I&E) £	1997/98 (I&E) £	1996/97 (I&E) £
Income/Receipts					
Subscriptions	9101	9152	10368	11021	11364
BCS services	278	278	167	0	106
Bank interest	1882	1882	2943	3322	3022
Vector advertising (incl. VAT)	2797	3047	2463	3471	4605
Other	550	366	264	108	351
Total receipts	14608	14725	16206	17922	19448
Expenditure/Payments					
Meetings	259	259	704	0	358
Administration	1227	1344	1423	1152	2135
BCS services	278	278	167	0	106
Vector production and despatch	15903	15802	17086	14584	15315
Education Vector supplement	-		-	108	3446
Projects	470	470	328	501	1117
Other	543	501	246	308	670
Total payments	18679	18653	19954	16653	23148
Assets summary:					
Bank and other balances		44984	49055	52525	52021
Debtors		858	733	2071	2188
Creditors		(4858)	(4776)	(5270)	(6151)
Net assets		40983	45012	49327	48059
Written off		(100)	(568)		

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 1999/2000, 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.

The sum written off comprises a cancelled invoice (Devon Systems)

Adjustments to R&P for 1999/2000 to obtain I&E.

Add to income:

Subscriptions	452.00 paid in 1998/99
	107.50 outstanding 1999/2000
	-508.50 advance payments for 1999/2000
Vector ads	-500.00 from Vol 15.4 (excluding VAT)
	750.00 outstanding for Vol 16.4 (excluding VAT)
Other	-135.00 cancelled subscription (Towers Perrin)
	-49.40 VAT refund, 1999/2000

Add to expenditure:

Vector	-3200.00 production of 15.4 (estimate) (actually 2914.91)
	83.33 credit note from 1998/99
	-832.31 late expenses claims, 1998/99
	3528.53 production of 16.4
	320.11 late expenses claims, 1999/2000
Admin	-51.70 late expenses claims, 1998/99
	168.73 late expenses claims, 1999/2000
Other	-31.85 VAT, 1998/99 Q4
	43.75 VAT, 1999/2000 Q4
	49.40 VAT due to BCS (1999/2000 Q3)
	-104.00 Vector storage 1998/99
	-104.00 Vector storage 1997/98
	104.00 Vector storage 1999/2000

Membership report

Nicholas reported on behalf of Rowena:

Membership at 30.4.00 (previous year's figures in parentheses)

	UK		FOREIGN		TOTAL	
	Number	Vectors	Number	Vectors	Number	Vectors
Sustaining	4 (4)	15 (12)	4 (4)	66 (76)	8 (8)	81 (88)
Corporate*	6 (8)	37 (57)	1 (3)	10 (30)	7 (11)	47 (87)
Corp. Ind*	20 (22)	24 (26)	2 (1)	2 (1)	22 (23)	26 (27)
Individual	130 (138)	128 (138)	218 (223)	218 (222)	348 (361)	346 (360)
Non-voting	15 (17)	15 (17)	0 (0)	0 (0)	15 (17)	15 (17)
Life	1 (1)	1 (1)	1 (1)	1 (1)	2 (2)	2 (2)
Library	1 (1)	1 (1)	6 (5)	6 (5)	7 (6)	7 (6)
Russians			10 (11)	10 (11)	10 (11)	10 (11)
APL Groups			13 (14)	39 (42)	13 (14)	39 (42)
						573 (640)

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

Further business

There were no questions to Officers nor any other business so the Chairman closed the meeting at approximately 2.15p.m. and handed over the meeting to Jon Sandles.

Anthony Camacho 20th May 2000

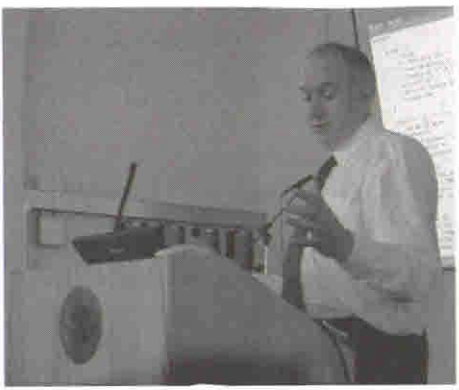
British APL Association: Vendor Forum May 19th 2000, following the AGM

reported by Jonathan Manktelow (jonathan@causeway.co.uk)

Dyadic

John Scholes took this opportunity to promote one of his favourite, and possibly one of the most under utilised features of Dyalog - dynamic functions.

His session was a whirlwind tour of the various techniques he uses to build powerful utilities built purely in dynamic functions. From the most simple - sink (a function that simply takes an argument and does nothing) all the way through to a powerful workspace difference tool, which produced a clear and comprehensive report of the differences between two workspaces.



The most striking thing about this presentation was that dynamic functions enable us to write procedural code in APL. Efficiently! Many classes of problem are much easier to solve using procedural and recursive code, than array manipulation. However using a lot of procedural looping code in APL can be quite slow when compared to the equivalent array based code. But because dynamic functions have a very simple syntax, the interpreter can execute them far quicker than the equivalent traditional APL.

The session clearly showed not only the power of dynamic functions, but also the fact that we do not have to throw away the procedural paradigm to enjoy the manipulation of matrices!

Soliton

Benoit Paquin, from Soliton, came to the conference to give most of us our first glimpse of the new SAX for Linux. This is the new Linux port of the Soliton APL interpreter. He quickly explained the pricing structure (the cheapest option being free for non-commercial use) and moved on to give us an overview of the structure of the system. The SAX development environment consists of a number of modules, with an APL interpreter running on a Linux machine at the core.

For most of his session Benoit concentrated on showing us how to build an APL powered calculator. With a Java user interface! This allows the front end of the calculator to be run on any Java enabled system that is connected to the Linux server.

After being shown how easy it was to move the simple user interface handling out to a Java GUI, but keep the power of a mainframe APL interpreter to process the data, it was not difficult to see how this technology could be used to produce very powerful multi-user APL systems, with good looking front ends, that will run on any machine attached to the network.

Causeway

The Causeway session started by showing some of the new features that have recently been added to CPro, Rain and NewLeaf, including drag and drop technology in CPro. Many of these new features will be explored in more detail at the Berlin conference at the end of July.

We then moved on to show a preview of GraPL, server edition (Graphing Power unleashed). This technology wraps the Rain graphics engine in an OCX, which allows users to embed quality publication graphics into systems developed in almost any programming tool under Windows.

To illustrate the power of this technology we showed a couple of web pages containing graphs generated on the fly by the web server. The server side code consisted of a few lines of VBScript, which could easily be extended to read data values from a database on the server before plotting the latest information.

Sustaining Members' News

Soliton Associates

Soliton is pleased to announce the availability of SHARP APL for Unix (SAX) version 6.0. SAX 6.0 comes in 2 editions: an Enterprise Edition (EE) and a Personal Edition(PE). Both editions of SAX are technically identical but the Enterprise Edition offers enhanced connectivity to Oracle databases and to OS/390 services such as DB2, VSAM and OS datasets accesses. The Personal Edition is only available for the Linux platform (RedHat, SuSe and Mandrake), is royalty free for non-commercial uses and can be downloaded from Soliton's FTP server. The Enterprise Edition is available for the IBM AIX, SunOS and Linux platforms and includes comprehensive support and update services.

SAX 6.0 EE will be available on September 4th while SAX 6.0 PE will be available from our FTP server in the course of August. More than 1000 pages of on-line documentation is included with both editions. Printed manuals can also be purchased separately.

SAX 6.0 offers improved performance over its predecessors, a high capacity Shared Variable Processor, a new file server, enhanced fonts and keyboard support. It has socket connectivity and excels as a server in a client-server architecture. In particular, the SAX 6.0 web server offers SSL encryption for secure transactions.

SAX 6.0 is the first version of SAX that runs under Linux and enables portability and scalability across all of its platforms; an application running on SAX 6.0 for Linux can be migrated to a SunOS or AIX environment at no cost (and vice versa). The low cost of entry offered by Linux will enable corporations to introduce SAX in their infrastructure while keeping their potential for growth through the scalability offered by SAX on its 3 supported platforms.

To ease development of SAX 6.0 applications, a new integrated development environment (IDE) is being developed and a first alpha release is intended to be available in August. The IDE is designed as a client and, being Java based, can be used across a network as, for example, a Windows client connected to a SAX server running under Linux.

Visit our web site: www.soliton.com/linux for more information about the Personal Edition of SAX 6.0. Enterprise Edition inquiries should be made to Laurie Howard (Europe) at +31 20 646-4475 and to Nancy Lamb (North America) at (716) 256-6466.

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

COMPLETE APL SYSTEMS

COMPANY	PRODUCT	PRICES(£)	DETAILS
Dyadic	IBM RS/6000 MD320	11,736	APL POWERstation (Greyscale) 27.5 MIPS, 7.4 Mflops RISC Processor 8Mb RAM, 120Mb Disk 19" 1280x1024 Greyscale Graph Display AIX, OSF Motif, Dyalog APL (1-user)
	IBM RS/6000 MD320	13,817	APL POWERstation (Colour) 27.5 MIPS, 7.4 Mflops RISC Processor 8Mb RAM, 120Mb Disk 16" 1280x1024 Colour Graphics Display AIX, OSF Motif, Dyalog APL (1-user)
	IBM RS/6000 MD320	22,656	Advanced APL POWERstation 27.5 MIPS, 7.4 Mflops RISC Processor 16Mb RAM, 320Mb Disk, 150Mb Tape 16" 1280x1024 Colour Graphics Display AIX, OSF Motif, Dyalog APL (1-user)
	IBM RS/6000 MD520	37,114	APL POWERsystem (8-users) 27.5 MIPS, 7.4 Mflops RISC Processor 16Mb RAM, 320Mb Disk, 150Mb Tape CD-ROM Drive, 16 Ports AIX, Dyalog APL (2-8 user licence)
	IBM RS/6000 MD530	72,054	APL POWERsystem (16-users) 34.5 MIPS, 10.9 Mflops RISC Processor 32Mb RAM, 1.34Gb Disk, 2.3Gb Tape CD-ROM Drive, 16 Ports AIX, Dyalog APL (8+ user licence)
	IBM RS/6000 MD540	122,842	APL POWERsystem (32-users) 41 MIPS, 13 Mflops RISC Processor 64Mb RAM, 1.7Gb Disk, 2.3Gb Tape CD-ROM Drive, 32 Ports AIX, Dyalog APL (8+ user licence)
Optima	IBM Compatible	poa	Complete networked or stand-alone solutions including configuration installation, maintenance and commissioning.

APL INTERPRETERS

COMPANY	PRODUCT	PRICES(£)	DETAILS
Beautiful Systems	Dyalog APL/W for Windows	poa	US Distributor of Dyalog APL products from Dyadic.
	Dyalog APL for Unix	poa	See Dyadic listing for product details.
The Bloomsbury Software Company			
	APL+PC Version 11	250	Upgrade to version 11 gives free runtime (£120 from any version)
	APL+Win v3.0	1350	A 32-bit Windows-hosted interpreter that runs under all Windows platforms including Windows 95. Note: Upgrade for £350 from version 1.8 or 2, £920 from version 1.0
	Migration to APL+Win	1060	from APL*PLUS/PC APL*PLUS/DOS any version.
			from earlier versions of APL*PLUS II
	APL+DOS	1250	APL*PLUS II DOS is renamed to APL+DOS.
	Migration to APL+DOS	760	from APL*PLUS/PC
	APL Link	200	Database Access
	APL Link Pro	500	
	APL*PLUS II for UNIX	poa	APL2000's 2nd generation APL for all major Sparc and Risc Unix workstations.
	APL*PLUS VMS	poa	2nd generation APL for DEC VAX computers under VMS.
	APL*PLUS Mainframe	poa	Enhances VS APL with many high performance, high productivity features. For VM/CMS and MVS/TSO offers simple upgrade from VS APL.
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.
Dyadic	Dyalog APL for DOS/386	995	Second generation APL for DOS.Runs in 32-bit mode, supports very large workspaces. Unique "window-based" APL Development Environment and Screen Manager. Requires 386/486 based PC or PS/2, at least 2Mb RAM, EGA or VGA, DOS 3.3 or later.
	Dyalog APL/W for Windows	995	As above, plus object-based GUI development tools. Requires Windows 3.0 or later.
	Dyalog APL for Unix	995-12,000	Second generation APL for Unix systems. Available for Altos, Apollo, Bull, Dec, HP, IBM 6150, IBM RS/6000, Masscomp, Pyramid, NCR, Sun and Unisys machines, and for PCs and PC/2s running Xenix or AIX. Oracle interface available for IBM, Sun and Xenix versions.
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.
	APL2 PC (US Version)	$630	Product No. 5799-PGG. PRPQ Number RJ0411. Order from 1-800-IBM-CALL
	APL2 PC (European Version)	£348	Product No. 5604-260. Part number 38F1753. From all IBM dealers, including MicroAPL.
	APL2 for OS/2 Entry Edition	$185	Part No 89G1556.
	APL2 for OS/2 Advanced Edition	$650	Part No 89G1697. Contains all facilities of the Entry Edition plus: DB2 interface; co-operative processing TCP/IP interface; tools for writing APs; TIME facility
	APL2 for Windows version 1.0	$1500	Product No. 5639-d46, part number 4229558.
	APL2 Runtime environment	$250	Part No. 39L8419 - Packager and runtime CDs. One additional runtime install $200, 5 additionals $900, 10 $1,500.
	APL2 for Sun Solaris	$1500	Product No. 5648-065.
	APL2 for AIX 6000	poa	Product No. 5765-012.
	APL2 Version 2	poa	Product No. 5688-228. Full APL2 system for S/370 and S/390
	APL2 Application Envt Vn2	poa	Product No. 5688-229. Runtime environment for APL2 packages
Insight Systems	Cognos/APL2000 Inc	poa	Leading distributor of APL2000 products in Denmark
	Dyadic Systems Ltd.	poa	Leading distributor of Dyalog APL products in Denmark
	IBM	poa	Leading distributor of IBM APL & GraphX products in Denmark
Iverson Software Inc.	J on the Web online registration ...		
	J Educational Edition	$95	
	J Standard Edition	$295	
	J Professional	$895	
	Books and accessories (discounts for reg users)		
	J Dictionary	$50	
	J User Manual	$50	
	J Phrases	$50	
	J Primer	$50	
	Concrete Math	$40	
	Exploring Math	$50	

	J User Conference Proceedings	$35	
	Mugs, T-shirts, Mousepads	$10 each	
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.
Lescasse Consulting	APL+PC	poa	Lescasse Consulting is the exclusive APL2000 distributor in France and also
	APL+Unix	poa	distributes in Switzerland and Belgium. Call for price quotes.
	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	APL.68000/X	1500-6000	Second-generation APL. Nested arrays, user defined operators, selective specification, etc. Multi-user AIX version with full OSF/Motif support.
	APL.68000 Level II Mac	520	Second generation APL. Full windowing interface.Hardware and software floating point support.
Oasis	Dyalog APL	poa	Dyadic Systems
	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	Fully fledged Windows development environment.
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.
	Dyadic and ISI products	poa	
	USA		
	Dyadic and ISI 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 Software	Adaytum Planning	poa	Full-featured Budgeting and Financial Planning system for medium to large enterprises.
APL Group (see Eventra)			
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@juno.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.
The Bloomsbury Software Company			
(for VSAPL)	Enhancements & Sharefile	poa	Component files, quad-functions & nested arrays for VSAPL under VM/CMS & MVS/TSO
	Compiler	poa	The First APL compiler!
(for APL2)	Sharefile/AP	poa	STSC's shared access component file system for APL2. Comparable to all APL*PLUS file systems: multi-user storage of APL2 arrays with efficient disk usage.
Causeway	*CausewayPro* for Dyalog/W	400/$600	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 (1.8 and above)
Cinerea AB	ORCHART	250	Organization chart package for IBM APL2/PC. Full & heavily commented source code included - free integration into other applications. NB: ASCII output with line-drawing (semi-graphic) characters for boxes.
CODEWORK	HELM	poa	Decision Support System for top management. Handles large multi-dimensional tables, data analysis, EIS presentations, generates HTML and Latex output. Platforms: MS-DOS, LAN, Windows 3.1/95/NT. Ideal for APL customisation (APL*PLUS II and Dyalog APL); more than 150 installed.
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.

Eventra	Qualedi	$1500-4000	Electronic Data Interchange (EDI) translation software for the PC, with strict compliance checking.
H.M.W.	4XTRA	poa	Front-end Foreign Exchange dealing / pos keeping
	Arbitrage	poa	Arbitrage modelling
	Basket	poa	Basket currency modelling
	Menu-Bar	poa	pull-down menu for APL*PLUS/PC
I-APL Ltd	Educational workspaces	5	PC format disks with the examples from: Thomson. Espinasse (Kits 1-4), Kromberg, Jizba & FinnAPL. All the examples to save your fingers!
IBM APL Products	A Graphical Statistical System	$250	for DOS, Product Number 5764-009
	(AGSS)	$500	for Workstations (OS/2, Aix, Solaris), Product Number 6764-092
		$2500	for CMS, Product Number 5764-011
INFOSTROY	APL*PLUS/Xbase Interface (I/386 Version 2)	$198	Complete package written in C. Comparable with the data, index & memo files of FoxPro, dBASE, & Clipper. Multi-user support. No DBMS license required.
	(DLL Version 1)	$198	The same in a DLL form! Gives your Windows applications all advantages of DLLs.
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	FRESCO Business Graphics	poa	Fast and easy business graphics DLL
	AP126/PC	poa	GDDM interface for Dyalog APL/W
	AP127/PC	poa	ODBC interface for Dyalog APL/W
	AP119/PC	poa	TCP/IP interface for Dyalog APL/W
	FACS	poa	EMMA-like interface to DB2 or ODBC databases

	ISAP	poa	MS Windows DLL for calling Dyalog APL/W from web pages via MS Internet Information Server. Visit www.lingo.com for a demo.
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
Ajay Askoolum	Consultancy	poa	APL+Win development and migration of actuarial, financial, mathematical applications.
Andrews	Consultancy	poa	APL programming and analysis, Year-2000 legacy systems, algorithms, tree-processing.
APL Solutions Inc	Consultancy	poa	APL systems design, development, maintenance, documentation, testing and training. Providing APL solutions since 1969.
AUSCAN Software	Consultancy	poa	APL software development, training
Bloomsbury Software	Consultancy	300-750+VAT	
Camacho	Consultancy	poa	Manuals; feasibility reports and estimates; analysis and programming; APL and MS Windows applications; Sharp, ISI APL, APL*PLUS, APL2/PC and other APLs spoken. Fixed price systems a speciality
Ray Cannon	Consultancy	poa	APL, C, Assembler, Windows, Graphics: PC and mainframe
Causeway	Consultancy and Training	poa	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.
Dinosoft Oy	Consultancy	poa	Specialised in very large databases.
Dyadic	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.
H.M.W.	Consultancy	poa	System design consultancy, programming. HMW specialize in banking and prototyping work.
Hoekstra Systems Ltd	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	Moving applications between platforms. Client/server development. Multilingual user interface.
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.
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.
George MacLeod	Consultancy	poa	Design and programming of new APL applications. Enhancing and maintaining existing APL applications. Porting existing APL applications from one APL system to another. Supporting users of APL applications. Experienced on both mainframe, UNIX and PC APL interpreters.

Mackay Kinloch Ltd	Consultancy	from £40/hr	Design, analysis and programming for banking, insurance and pensions, financial planning and modelling, corporate performance and legal reporting
MicroAPL	Consultancy	poa	Technical & applications consultancy.
Millnta 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	250-500	Business Forecasting & APL Systems.
Oasis	Consultancy	poa	Expertise in APL system design, Project management, conversion, migration, tuning; for all APL versions (10+ years experience)
Object Oriented Ltd	Consultancy	poa	General APL consulting, code recycling – mainframe to PC, performance tuning.
Omega Computing	Consultancy	poa	APL consultancy, programming, etc.
Optima	Consultancy	poa	A range of consultants specialising in all areas of pharmaceutical, industrial and financial systems with 5-15 yrs experience on both PC and mainframe.
RadSys Technologies	Consultancy	poa	Areas of expertise: financial systems, risk analysis systems, healthcare systems.
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.
Rex Swain	Consultancy	poa	Independent consultant, 20 years experience. Custom software development & training, PC and/or mainframe.
Rochester Group	Consultancy	poa	Specialise in MIS using Sharp APL
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.
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.
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.
Bloomsbury Software	Training	poa	Contact the company for details.

ComLog	Comic-Logger	$25.95+p&p	APL*PLUS II comic-book inventory system. Shareware version available on America OnLine.
HMW	Employment	poa	Contractors and permanent employees placed.
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 100612.1057@compuserve.com.
Oasis	Training	poa	Introductory courses in APL Advanced courses for different APL versions
Renaissance Data Systems	Booksellers		The widest range of APL books available anywhere. See Vector advertisements.

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 Club Germany	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)
Capital PCUG	Washington, D.C.	Monitor	Monthly meetings, occasional classes	free
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	SEAPL Newsletter	Quarterly meetings	$10
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	Gimme Arrays!	Monthly Meetings, APL skills database, J SIG, Toronto Toolkit	$25

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	Marienhoehe 86, 25451 Quickborn, Germany. Tel: +49 4106 60977 Fax: +49 4106 67869 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: adsys@ibm.net
Adaytum Software	Douglas Rowley	13 Great George Street, BRISTOL BS1 5RR, UK. Tel: 0117-921 5555
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: 100331.644@compuserve.com
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 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 Club Germany	Dieter Lattermann	Rheinstraße 23, D-69190 Walldorf, Germany. Tel: +49 6227-63489 Compuserve: 100332,1461
APL Group (see Eventra)		
APL Software\Services	Dick Holt	3802 N Richmond St, Suite 271, Arlington, VA 22207 USA Tel: +1 (703) 528-7624; Fax: +1 (703) 528-7617; Email: dick.holt@juno.org
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: elandau@cais.com
Ajay Askoolum	Ajay Askoolum	42 Hanworth Road, Redhill, Surrey RH1 5HT Tel: 01737-771643 Email: ajay@askoolum.freeserve.co.uk
Association Francophone pour la promotion d'APL	Ludmila Lemagnen	174 Boulevard de Charonne, F-75020 Paris, FRANCE Email: lemagnen@aol.com
AUSCAN Software Ltd	Richard Procter	PO Box 39, Mansfield, Ontario L0N 1M0 Canada Tel: +1-705-434-1239 Email: rjp@interlog.com
BACUS	Joseph De Kerf	Roolnberg 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
Bloomsbury Software	Peter Day	Bloomsbury House, 74-77 Great Russell St., London WC1B 3DA, UK. Tel: +44(0)20 7436 9481 mobile +44(0)836 254660 Fax: +44(0)20 7436 0524 mobile +44(0)385 850629 Email: pd@bloomsbury-software.co.uk
Camacho	Anthony Camacho	11 Auburn Road, Redland, Bristol BS6 6LS, UK. Tel: 0117-973 0036. email: acam@tesco.net
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: causeway@compuserve.com
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	1, Heifer Mill Cottages, Mosterton, Beaminster, Dorset DT8 3HG, England. Tel: +44 (0)7931 370304. Email: earthspot@aol.com
CODEWORK	Mauro Guazzo	Corso Calroll 32, 10123 Torino, Italy. Tel: +39 11 885168 Fax: +39 11 812 2652 Email: codework@inrete.it
ComLog Software	Jeff Pedneau	18726 Bloomfield Road, Olney, MD 20832 USA Tel: +1 (301) 260-1435 Email: jeff@softmed.com
CPCUG	Lynne Sturtz	Capital PC User Group, 51 Monroe Street, Suite PE-2, Rockville, Maryland 20850-2421, USA. Tel: +1 (301) 762-9372 Fax: (301) 762-9375.
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
Dutch APL Association	Bernard Smoor (Sec)	Postbus 1341, 3430BH Nieuwegein, Netherlands. Tel: +31 347 342 337 Fax: +31 347 342 342
Dyadic Systems Ltd.	Peter Donnelly	Riverside View, Basing Road, Old Basing, Basingstoke, Hants RG24 0AL, UK. Tel: 01256-811125 Fax: 01256-811130
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
Eventra	Stuart Sawabini	Merritt Crossing, 440 Wheelers Farms Road, Milford, CT 06460, USA. Tel: +1(203) 882-9988. Fax: +1 (203) 882-9946 Email: ssawabini@eventra.com; eshaw@eventra.com
Evestic AB	Olle Evero	Berteliusvagen 12A, S-146 38 Tullinge, Sweden Tel&Fax: +46 778 4410 Email: olle.evero@mailbox.swipnet.se

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 749328 Fax: 01622 749365 E-mail: martin@gsoft.freeserve.co.uk
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
H.M.W.Trading Systems	Chris Hogan	Hamilton House, 1 Temple Avenue, Victoria Embankment, 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 Email: bob.hoekstra@khamsin.demon.co.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: 100612.1057@compuserve.com
IBM APL Products	Nancy Wheeler	APL Products, IBM Santa Teresa, Dept REN/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	Alexei Miroshnikov	3 S. Tulenin Lane, St. Petersburg 191186 Russia. Tel:+7 812 312-2673 Fax:+7 812 311-2184 Email:aim@infostroy.spb.su
Insight Systems ApS	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
Iverson Software Inc.	Eric Iverson	33 Major Street, Toronto, Ontario, Canada M5S 2K9. Tel: +1 (416) 925-6096; Fax: +1 (416) 488-7559 Email: info@jsoftware.com
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 Build.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	Flat 3, 42 Queen Annes Road, Bootham, YORK YO30 7AF Tel: 01904-651544 Email: joho@jaustria.freeserve.co.uk
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	Steven J Halasz	1105 Chicago Avenue, Suite 155, Oak Park, IL 60302, USA. Tel:+1 800 546 4621-1 Email: sjh@sjhalasz.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)131 228 5235 Email: akinloch@globalnet.co.uk
George MacLeod	George MacLeod	37 Newhouse Rd, Bovingdon, Herts, HP3 0EJ, UK. Tel: 01442-834015 Email: GeorgeMacLeod@simcorp.com
Mercia Software Ltd.	Gareth Brentnall	Holt Court North, Heneage Street West, Aston Science Park, Birmingham B7 4AX, UK. Tel: 0121-359 5096. Fax: 0121-359 0375
MicroAPL Ltd.	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	4215 St-Andre, Montreal, CANADA H2J 2Z3. Tel: +1 (514) 529-1434 Email: danb@dsuper.net; milinta@epost.ca
Ellis Morgan	Ellis Morgan	Myrtle Farm, Winchester Road, Stroud, Petersfield, Hants GU32 3PE, UK. Tel: 01730-263843 Email: Ellis@mrtlfrm.demon.co.uk
NY Sig	David Siegel	PO Box 2697; New York, NY10163-2697, USA. Email: NYSIGAPL@ACM.ORG
Oasis b.v.	Theo Zwart, Louis Rijkse	Lekstraat 4, 3433 ZB Nieuwegein, Holland. Tel: +31 30 60 66 336 Fax: +31 30 60 65 844 Email: info@oasis.nl or rijkse@oasis.nl

Object Oriented Ltd	Walter G. Fil	Am Grendel 2, CH-6004 Luzern, Switzerland. Tel: 41 41 418 70 70 Fax: 41 41 418 70 77 Email: info@object-oriented.com
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	115 Brighton Road, Purley, Surrey CR8 4HE, UK. Tel: +44 (0)20 8763 2490 Fax: +44 (0)20 8763 2491 Email:mailbox@optima-systems.co.uk
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 313, Newtown, CT 06470, USA. Tel: +1 (203) 270-9729 Email: rendata@aplbooks.cnchost.com or orders@aplbooks.cnchost.com
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
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, 00100-Roma Trullo, Italy Email: marsac@vnet.ibm.com
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	Laurie Howard	Soliton Associates Ltd, Groot Blankenberg 53, 1082 AC Amsterdam, Netherlands Tel: +31 20 646 4475 Fax: +31 20 644 1206 Email:sales@soliton.com
SovAPL Russian Chapter of SIGAPL	Alexander Skomorokhov	PO Box 5061, Obninsk-5, Kaluga Region 249020, Russia Tel: +7(08439)31463 Fax: +1 (530) 504 8194 Email:askom@obninsk.com
Strand Software Inc	Anne Faust	19235 Covington Court, Shorewood MN 55331 USA Tel: +1 (612) 470-7345 Email: sales@jsoftware.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) 222-2759 Fax: +1 (818) 222-9250
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@cwcom.net
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/
Adaytum Software	www.adaytum.com/
AFAPL	www.ensmp.fr/~scherer/langlet/ (Journal available on line)
APL2000	www.APL2000.com/
APL-385	www.demon.co.uk/apl385/
APL Journal, Germany	www.rhombos.de/rb/apljourn.htm
AUSCAN	www.interlog.com/~rjp/auscan/
Eke van Batenburg	wwwbio.LeidenUniv.nl/~Batenburg/index.html
Bloomsbury	www.bloomsbury.co.uk/software
Capital PC User Group	http://cpcug.org/
Causeway	www.causeway.co.uk/
CODEWORK	www.inrete.it/cdwk/eng/homee.html
COSY (Bob Armstrong)	www.cosy.com/
Dinosoft Oy	www.dinosoft.fi/
DMOZ - Open Directory	http://dmoz.org/Computers/Programming/Languages/APL/
Dyadic Systems Ltd	www.dyadic.com/
DynArray	www.dynarray.com/
Eventra	www.eventra.com/
FinnAPL	www.pyr.fi/apl/
Godin London Inc	www.godin.com/
Hoekstra Systems	www.khamsin.demon.co.uk/about/hsl/noframe.html
IBM APL2	www.ibm.com/software/ad/apl
Infostroy	www.insight.dk/infostroy/
Insight Systems ApS	www.insight.dk/
Iverson Software Inc	www.jsoftware.com/
Japan APL Association	www.naska.co.jp/JAPLA/
Lescasse Consulting	www.lescasse.com/
Lingo Allegro USA Inc	www.lingo.com/
Mackay Kinloch	www.users.globalnet.co.uk/~akinloch/akinloch.htm
MicroAPL Ltd	www.microapl.co.uk/
Milinta Inc	www.dsuper.net/~danb/milinta
Oasis b.v.	www.oasis.nl/
Optima Systems Ltd	www.optima-systems.co.uk
Renaissance Data	www.aplbooks.cnchost.com/
RE Time Tracker Oy	www.rett.fi/
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.jsoftware.com/
Rex Swain	www.rexswain.com/
Toronto SIG (for Toolkit)	www.torontoapl.org/
Jim Weigang	www.chilton.com/~jimw/
Weighahead Systems	www.weighahead.com

Zark Newsletter Extracts

introduced by Jon Sandles

I hope you are all still enjoying these Zark reprints! Once again, we include a new crossword for your enjoyment and another classic APL problem. It would be interesting to see some new solutions to these problems, perhaps using Dyadic's dynamic functions, or some J or even K. Solutions will be printed in the next issue.

LIMBERING UP:
Matrix Searching

(The purpose of this column is to work some flab off your APL midsection. Like muscles, your APL skills can atrophy if not exercised with adequate frequency and variety. This column presents a task for you to perform. Set aside a few minutes from your busy schedule and work the task. Mail in your solution and stay tuned for the results.)

Dyadic ⍳ ("index of") is the principal searching function in APL. The value of each of the elements in its right argument is searched for in the vector left argument. The left argument *must* be a vector. The result is the same shape as the right argument and contains indices into the vector left argument where the elements of the right argument are first located.

Not all problems involve searching for values in a vector. Many searching problems involve matrices. For example, given a ten-row, eight-column character matrix of eight-character identification codes for ten products, locate each of the ten products in a master list of 1000 products (a 1000 by 8 character matrix). If dyadic ⍳ worked on matrices as well as vectors, the solution would be straightforward:

```
INDICES←MASTERLIST ⍳ PRODUCTCODES
```

Where *MASTERLIST* and *PRODUCTCODES* are character matrices with shapes `1000 8` and `10 8` respectively, and *INDICES* is a ten-element vector of row indices into *MASTERLIST* where each row of *PRODUCTCODES* is first located.

Unfortunately, dyadic ⍳ requires a vector left argument. Given a matrix, it signals *RANK ERROR*. Until ⍳ is extended to work for matrices as well as vectors (we're optimistic the day will come), alternative algorithms need to be used. Typically,

these alternative algorithms hide in sub-functions that have the same syntax as dyadic ⍳ and that go by the name *IOTA*, *MIOTA*, *CMIOTA*, *LOOKUP*, *ROWFIND*, etc. Some implementations of APL even come equipped with a specially compiled or assembled function that serves this purpose.

Many different APL algorithms exist for searching character matrices. Your task is to compose the fastest ones you can muster and mail them in. The syntax should be:

```
INDICES←CMATL CMIOTA CMATR
```

CMIOTA should work on character matrices. Searching problems that involve numeric matrices are relatively rare. If your algorithm works on numeric matrices too, that's nice, but you get no extra credit!

The quality of the different algorithms will be measured based upon *speed* alone. However, all submitted functions will be tested on a variety of different implementations, and for different argument sizes. We'll try them with left arguments having 2, 8, 64, 256, and 2048 rows, right arguments having 2, 8, 64, 256, and 2048 rows, and for 2, 4, 8, and 16 columns.

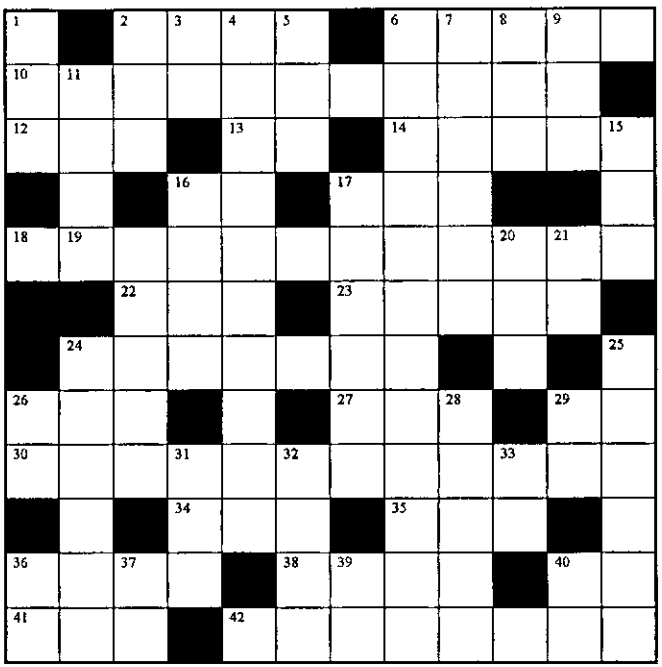
Please send your solutions to Vector Production at:

Vector Production
Brook House
Gilling East
YORK YO62 4JJ
UK

Reprinted with kind permission from Zark APL Tutor News, a quarterly publication of Zark Incorporated, 23 Ketchbrook Lane, Ellington, CT06029, USA

Crossword

Crossword



Across

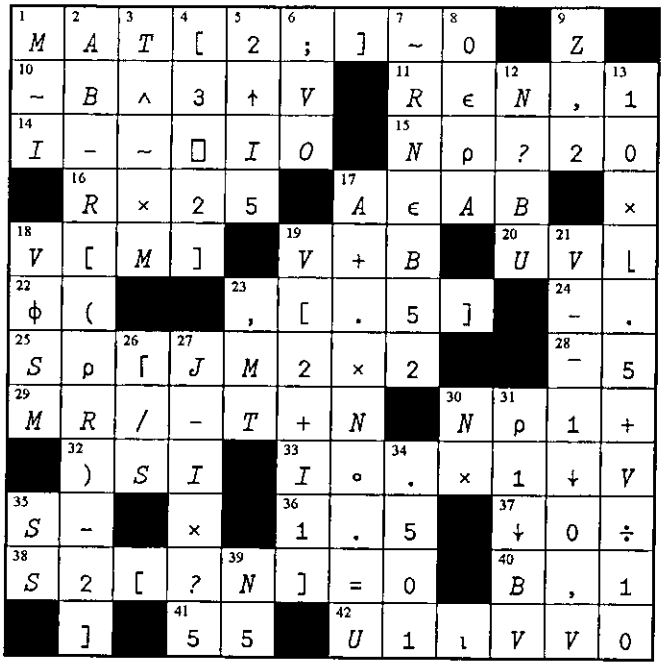
2. The shorter of ⍳*B* or ⍳*D*
6. (*A*-⌊*A*)
10. Round each element of P to the nearest tenth
12. Flag the rows of two-column N for which (⍋*N*[*I*;])≡⍒*N*[*I*;]
13. Right argument of ? when dealing a poker hand
14. *BLT*[1]-*BLT*[2]+*BLT*[3]-*BLT*[4]+*BLT*[5]-...
16. Add K to the first element of the array S. Use the result as the number of counting numbers from which to select A different random numbers. Add R to these random numbers but don't allow any to get larger than B. Multiply each resulting number by 5 and use these numbers to fill an array having shape D52. (Put the resulting expression in 5D, 34A, 3D, 16A, 36D, 39D, 29A and 25A.)
17. The square root of one-third of AD (first three symbols only)
18. Future value of 1 per year for N years at accumulation rate I (where A=1+I)
22. (+*R*9⍴*R*9≠*R*9)--⎕*IO* for scalar R9

23. `(1 0 ¯1)[¯1 0 1⍳×ASI]`
24. Future value of 1 in N years at interest rate I
26. `''⍴((V=2)/⍳⍴V),⎕IO+⍴V`
27. `1⌽⌽')N('`
29. See 16 across
30. Probability of "four of a kind" given five cards from a deck of 52
34. See 16 across
35. `9,(A-1)⍴0`
36. Add the scalar A to the singleton T `(1∧.=⍴T)` returning a 1-element vector
38. `(-V)+⍳K` for $K \geq V$
40. Row-wise addition
41. A random integer between 1 and 25
42. Value that compounds in N years to 1 at interest I

Down

1. Function used to match a vector to the rows or columns of a matrix
2. `(N≠0)⍴⎕IO` for non-negative scalar N
3. See 16 across
4. The quotient of 191 and 45, rounded to the nearest integer
5. See 16 across
6. Present value of 1 per year for N years at the accumulation rate A (A=1÷interest rate)
7. `DNA[;,⍉(10,1+⍴DNA)⍴⍳1+⍴DNA]`
8. $B \neq 0$ for numeric B
9. A peculiar language
11. If TA is a scalar, ravel it; otherwise return TA
15. T random numbers from the set `⍳I`, without repeats
16. Equivalent of `(⍴R)⍴` on a numeric scalar without using `⍴` (see 40D, 16A)
17. `(*N+⊗A)`
19. `¯I23+1` in origin 0
20. Now, what led to this error?
21. Present value of 1 per year forever, at interest rate I
24. `2+⍳3` in either origin
25. The first S elements of `N[25/⍳1+⍴N;]`
26. `')V('~(⍷''')~''')~'~'`
28. `((5-⍴V)+V)` where $(\rho V) \geq 5$
29. `+/+\5/1`
31. `(⍴T)≠0` for vector T
32. `8 16 24 32 40 ...` `8×I` in origin 1
33. `×/⍳9` in origin 1
36. See 16 across
37. `1⍴5`
39. See 16 across
40. The two simplest inverse functions

Solution to Crossword in 16.4



J-ottings 25: The I-spy book of J

by Norman Thomson

Dear J-ohn and J-anet,

How would you like to be a security man or woman when you grow up? It's a very important job these days on account of the enormous volumes of personal and business data which fly through cyberspace every microsecond. If this is a career which attracts you, have you considered telling your teacher what a very good medium J is for getting started in this area?

Here are a few things you should know before we find you a uniform. First you should appreciate that cryptographic systems divide broadly into two categories, namely those based on *transposition* and those based on *substitution*. (I'm afraid you will have to ask your Mummy or Daddy to explain what these big words mean.) In practice many current coding systems involve both of these techniques. Your first lesson will concentrate on a subset of the second of these subdivisions, that is on those in which characters are first converted to numerals and then replaced by *ciphers*. The systems concerned work to a general pattern in which there are *keys* of two kinds, *public* and *private*. I make a public key freely available to anyone who wants to send me *enciphered* messages. The relative security of any cryptographic system is proportional to the time taken by the "enemy" (that is the hackers) to determine a *private* key which allows me, and only me, to *decipher* messages. I am aware of course that the enemy will *analyse* my messages in order to try and break the code by discovering my private key. This is the process which is known as *cryptanalysis*.

The basics of such methods are simple, and J is great for describing them. My first illustration concerns multiplicative codes which depend on the clock arithmetic which you do at school. As you know, the world of sums contains only positive integers and small ones at that. Should any of these accidentally get too big for their boots, they are simply trimmed down to size by taking away the clocksize. And should one of them stray into naughty negative regions then adding the clocksize (`cs`) an appropriate number of times is all that is needed to bring it back into the orderly region of `i.cs` .

The essence of clock multiplication is the remainder verb `|` applied to a table based on `i.`

```
   cmultab=.|*/~@i.          NB. clock multiplication table

   cmultab 5
0 0 0 0 0
0 1 2 3 4
0 2 4 1 3
0 3 1 4 2
0 4 3 2 1
```

(If you don't like the column and row of zeros just drop them:

```
   cmtab=.| */~@ }.@:i.

   cmtab 5
1 2 3 4
2 4 1 3
3 1 4 2
4 3 2 1   )
```

The *inverse* of a clock number x is that value y for which xy=1 in clock arithmetic, so that for cs=5 the tables above show that 1 and 4 are self-inverse, and that 2 and 3 are each the inverse of the other.

Now assign a different clocksize :

```
   cs=.26
```

From this you can guess that I have an alphabetic message in mind, with letters translated into numbers in the obvious way, A=1, B=2, etc. Encryption consists of multiplying each message number by the public key, (that is one of the numbers between 1 and 25 which has no common factor with 26) and then simplifying this by clock arithmetic

```
   enc=.cs&|@*                    NB. x.=key, y.=number
   5 enc 22 5 3 20 15 18          NB. encrypt "VECTOR"
6 25 15 22 23 12
```

To decipher this coded message I need to repeat this process, only now using the *inverse* of 5. The restriction put on the key in parentheses above guarantees that such a number will exist and be unique. To find the *multiplicative inverse* of a key with respect to cs, multiply the key by all the integers in the field and perform clock arithmetic using the key. The inverse is the index of whichever of these values is one:

```
   minv=.i.&1@(]|(*i.))           NB. syntax is 'key minv field'
   5 minv 26
21
```

Decipherment of "VECTOR" as coded above is simply a further encryption using the inverse key:

```
   21 enc 6 25 15 22 23 12
22 5 3 20 15 18
```

I make 5 known to all my correspondents as a public key, and hence implicitly to the enemy, who is presumed to be smart enough to work out the broad method, but needs to know cs in order to discover the inverse key which turns code back into plain text. It would not be a very difficult exercise to find these quantities using the parameters above, however by using a much larger cs, and redefining the encrypting verb so that letters are dealt with in blocks, a modest degree of security can be achieved:

```
   cs=.2752
   enc=.cs&|@*

   379 enc 2205 320 1518      NB. key=379
1839 192 154
   379 minv cs                NB. multiplicative inverse of 379
1779
   1779 enc 1839 192 154      NB. decipher coded message
2205 320 1518
```

One step which could be made towards greater security is to use an *exponential* cipher rather than a multiplicative one, that is instead of multiplying the code by the key, it is raised to the *power* of the key. Uniqueness of inverse requires that cs be a *prime number*. Otherwise the only change in J terms from * to ^ in enc :

```
   cs=.29
   15(cs&|@^)22 5 3 20 15 18
0 10 11 0 0 0
```

The zeros in the above indicate that there is a problem, namely that numbers such as 15^{22} are very large and exceed the capacity of the computer. This is easily solved since

(i) exponentiation is just repeated multiplication, and

(ii) multiplication in clock arithmetic follows the rules of multiplication in ordinary arithmetic, that is if a and b are the values of A,B and C when reduced to clock integers, then ab, if necessary reduced to a clock integer, is equal to the clock integer reduction of AB. (In mathematical terminology a=A(mod n) and b=B(mod n) implies that ab=AB(mod n)).

So define clock multiplication:

```
mul=.cs&|@*
```

and insert this into "key" replicates of the code:

```
   eenc=.mul/@#
   5 eenc &> 22 5 3 20 15 18       NB. encrypt "VECTOR", key=5
13 22 11 24 10 15
```

The & conjunction (&> is equivalent to "each") is necessary due to the non-scalar nature of the verb `mul` .

For the purposes of decipherment, mathematics dictates that the inverse key is the multiplicative inverse of one less than `cs`:

```
   5 minv 28                       NB. multiplicative inv. of 5
17
   17 eenc&> 13 22 11 24 10 15     NB. decipher message
22 5 3 20 15 18
```

This improves security a bit, but not by an enormous amount since an enemy with computers at his disposal would not take long to work out `cs` and hence, given that my key is public knowledge, to work out `inv cs`. A cryptographer's Holy Grail is to find a way in which to make his key completely public so that anyone can send him messages, while at the same time making the rule for computing the decipherment key so complex that the enemy has little hope of finding it, however massive the computing power he has available.

This remained an open problem in the world of cryptography until 1977 when a major breakthrough was achieved through the invention by of the so-called RSA ciphers at the Massachusetts Institute of Technology. These are named after the initials of their inventors R.L. Rivest, A. Shamir and L. Adelman. Their idea was that `cs` should be the *product* of two primes, say 3551=53*67, following which any public key must then be coprime to (53-1)*(67-1)=3432, which is also the number used to calculate the multiplicative inverse. Choosing 191 as the key, and resetting the `eenc` verb gives:

```
   cs=.3551
   eenc=.(3551&|@*)/@#

   191 eenc&> 2205 320 1518        NB. encipher "VECTOR", key=191
489 2774 2274
   191 minv 3432                   NB. multiplicative inv. of 191
575
   575 eenc&> 489 2774 2274        NB. decipher message
2205 320 1518
```

Of course the primes used in the above illustration are very small. In practice two very large prime numbers, say of the order of 10^{200}, would be chosen. Factoring products of this size is a very hard problem given the present state of the mathematical and computational arts, and so what is available to me is a public key which I can broadcast to everybody, but for which, provided I keep the two prime factors a secret, I have a private key which ensures that only I can decipher my incoming messages.

Now, dear J-anet and J-ohn, just think how little programming you have had to do to take you from the clock arithmetic which you love to techniques which are the basis of the day-by-day encryption of millions of business and financial transmissions. Will it by any chance be one of you who cracks the factoring algorithm? (For the answer, see Vector volume 99 no. 4).

Magic Squares

by Ian Clark

In the geocentric cosmology of Ptolemy, the classical seven planets (which include the sun and moon, but don't include any beyond Saturn because they hadn't been discovered) each sit more or less tightly on their own steadily rotating crystal sphere surrounding the earth. The fixed stars are attached to an outermost sphere. Amazingly all the spheres share roughly the same rotational axis, which causes the planets to confine themselves to a narrow band of fixed stars called the *zodiac* (a Chaldean word). The further away the planets are, the slower they creep round the zodiac, with dull old Saturn being the slowest of all. In fact they can be ranked from the slowest to the fastest like this:

```
SERIES←'SATURN' 'JUPITER' 'MARS' 'SUN' 'VENUS' 'MERCURY' 'MOON'
```

a fact known from antiquity. Modernists who can't think back past the heliocentric theory might care to note that if 'EARTH' is substituted for 'SUN', and omitting 'MOON' (which even the heliocentric theory recognises as going round the earth) then this ancient SERIES describes the planets in reverse order of mean distance (d) from the *sun*. Furthermore, d increases with orbital period (t) according to the formula d3=kt2. This is the Third Law of Johannes Kepler (1571–1630), who predated Isaac Newton (1642-1727) by over a century.

Do you know how the days of the week got their names? In ancient times the Chaldeans actually assigned them to planets in the following sequence, based on their order in SERIES:

```
      (7 3⍴SERIES)[;1]
SATURN  SUN  MOON  MARS  MERCURY  JUPITER  VENUS
```

The Chaldeans, not possessing computers, or at least computers running APL, employed the ancient sacred geometric figure known as the *heptagram* to compute the above expression. The original planetary god-names are most clearly seen in French, Spanish and Italian names for the days of the week, but if you map the Roman gods onto corresponding Germanic gods, you can see it in English too. Thus: Mars = Tiw (war god), giving Tuesday; Mercury = Woden (smart god), giving Wednesday; Jove = Thor (thunder god), giving Thursday; Venus = Freya (love goddess), giving Friday; and Saturn, Sun and Moon of course giving Saturday, Sunday and Monday respectively.

Not only did the planets get their very own days of the week in antiquity, but they also got their very own magic squares. Seven magic squares of orders 3 thru 9 were assigned to the planets in terms of the rate they orbited round the zodiac (i.e. in the sequence of SERIES), thus Saturn got the smallest order (3) and Moon got the largest (9). Here they are:

```
SATURN JUPITER      MARS            SUN
4 9 2   4 14 15  1  11 24  7 20  3   6 32  3 34 35  1
3 5 7   9  7  6 12   4 12 25  8 16   7 11 27 28  8 30
8 1 6   5 11 10  8  17  5 13 21  9  19 14 16 15 23 24
       16  2  3 13  10 18  1 14 22  18 20 22 21 17 13
                   23  6 19  2 15  25 29 10  9 26 12
                                   36  5 33  4  2 31

VENUS                   MERCURY                 MOON
22 47 16 41 10 35  4    8 58 59  5  4 62 63  1  37 78 29 70 21 62 13 54  5
 5 23 48 17 42 11 29   49 15 14 52 53 11 10 56   6 38 79 30 71 22 63 14 46
30  6 24 49 18 36 12   41 23 22 44 45 19 18 48  47  7 39 80 31 72 23 55 15
13 31  7 25 43 19 37   32 34 35 29 28 38 39 25  15 48  8 40 81 32 64 24 56
38 14 32  1 26 44 20   40 26 27 37 36 30 31 33  57 17 49  9 41 73 33 65 25
21 39  8 33  2 27 45   17 47 46 20 21 43 42 24  26 58 18 50  1 42 74 34 66
46 15 40  9 34  3 28    9 55 54 12 13 51 50 16  67 27 59 10 51  2 43 75 35
                       64  2  3 61 60  6  7 57  36 68 19 60 11 52  3 44 76
                                               77 28 69 20 61 12 53  4 45
```

A magic square of order n is an arrangement of the first n^2 natural numbers in an n-by-n square, such that each of the n rows, n columns, and 2 main diagonals, sums to the same total: m. If you look at the function listing in the appendix, you see that the function: magic (which returns 1 if a given matrix is a magic square, 0 otherwise) computes these totals in order to check they are all the same.

Without knowing anything else about the magic square, what must the number m be? Well, since there are n rows, each totalling m, and furthermore the grand total of the whole square is the sum of the first N natural numbers (½N(1+N)), where $N= n^2$, it follows that (in ordinary arithmetic notation):

$mn = ½ n^2 (1+n^2)$

or, dividing through by n:

$m = ½n(1+n^2) = ½(n+n^3)$

which in Dyalog version 8.2 can be defined as an in-line function like this:

```
total←{0.5×⍵+⍵*3}
```

The main use of magic squares from the Chaldeans onwards was in the drawing of talismans. Thus to inscribe the matrix SUN say, was to enshrine in the talisman the special virtues of the sun, namely: strength, majesty, imperium, etc. But in place of the whole magic square, you could use just the grand total, called its *magic number*. These numbers have an important significance in the occult art known as *gematria*. For example, summing along each row of SUN we get:

```
      +/SUN
111 111 111 111 111 111
```

and summing the row sums themselves we get the grand total, or magic number:

```
      +/+/SUN
666
```

...the number of the Beast! Would you have guessed it had anything to do with the sun?

It seems these seven magic squares were passed on down through the ages in exactly the same layout, so that occultists talk of *the* magic square of order 7, or *the* magic square of Venus. But are these the only magic squares of a given order? Of course not. Clearly you can transpose a magic square and this preserves the magic property, since rows become columns and vice-versa, but more importantly the two diagonals are mapped onto themselves. Thus:

```
      magic ⍉VENUS
1
```

You can also find permutations of rows and columns which preserve the magic property. Applying the same permutation to rows and columns will map the first diagonal onto itself, but if the permutation is its own mirror image this preserve the second diagonal also. One such permutation is (⌽⍳7): In fact all rotations, transpositions and reflections will work, because the contents of any row, column or diagonal keep together, so they will always sum to the value they did before.

```
      (⌽⍳7)⌽[2] VENUS
28  3 34  9 40 15 46
45 27  2 33  8 39 21
20 44 26  1 32 14 38
37 19 43 25  7 31 13
12 36 18 49 24  6 30
29 11 42 17 48 23  5
 4 35 10 41 16 47 22
      magic (⌽⍳7)⌽[2] VENUS
1
```

Notice too that you can make an essentially new magic square by replacing the highest number (49) with 1, the next highest with 2, and so on. In fact this is equivalent to calculating (50-VENUS), and so we can verify:

```
      magic 50-VENUS
1
```

Does this work with any magic square (using $1+n^2$ in place of 50)? Yes. We'll prove it by using APL notation itself. (How often do you see APL used for mathematical proofs? Why not? It's a formal language, isn't it?)

Let n be the order of the magic square (n=7 for VENUS) and let a be any vector of n elements summing to the row-total (call it: N) So:

```
N=+/a
```

Now replacing every element `a[i]` of `a` with `(1+n*2)-a[i]`, and summing it all, is equivalent to the expression:

```
+/(1+n*2)-a
```

The subexpression in brackets, `(1+n*2)`, is scalar, but combining it with `a` makes it behave like a vector of `n` equal elements. So, by simple arithmetic operations, the following equivalent lines can be successively derived:

```
+/(1+n*2)-a
(+/nρ(1+n*2))-(+/a)
(n×(1+n*2))  -(+/a)
(n+n*3)      -(+/a)
(2×N)        -N
N
```

You'll recognise the bracketed subexpression `(n+n*3)` in the last line but-one as equalling `2×N` from the definition of fn: `total`. Since this applies to any vector of n integers summing to `N`, it applies to every row, column and diagonal of any given magic square of order `n`.

But let us ask: how many distinct magic squares are there of order n, if we don't distinguish row/column permutations, rotations or reflections of a given square? For n=3 there is only one, namely SATURN. To see this, run the fn: comb with arg 3 (the order of the magic square). A listing of: comb is appended to this article. Fn: comb returns a set of Boolean vectors, each designating a choice of 3 numbers from `ι3*2` which happens to sum to `(total 3)`. All such vectors are found. If

you want to see the numbers this corresponds to, run fn: ros on the result of comb.

```
      ↑comb 3
1 0 0 0 1 0 0 0 1
1 0 0 0 0 1 0 1 0
0 1 0 1 0 0 0 0 1
0 1 0 0 1 0 0 1 0
0 1 0 0 0 1 1 0 0
0 0 1 1 0 0 0 1 0
0 0 1 0 1 0 1 0 0
0 0 0 1 1 1 0 0 0
      ros comb 3
1 5 9  1 6 8  2 4 9  2 5 8  2 6 7  3 4 8  3 5 7  4 5 6
```

You see that there are only 8 possible distinct *magic triples* (let's call them) – combinations of numbers which add to (*total* 3) = 9. Yet SATURN contains 8 distinct magic triples, 3 rows, 3 columns and 2 diagonals. So SATURN contains every possible magic triple.

In my university course on Computing I occasionally used this fact to cause a student with no intellectual pretensions to win a best-of-three contest against one of the self-acknowledged brightest students in the class (to the latter's exquisite chagrin!). I had them 'volunteer' to challenge each other to 'Number Scrabble', a game played with 9 counters numbered 1 to 9 placed face-up between two players. Each player take one counter in turn. The aim is to be the first to exhibit a selection of three counters summing to 15. Deceptively simple. Here's a sample game to illustrate some of its subtlety:

1. Player 1 takes 5 holds: 5
2. Player 2 takes 2 holds: 2
3. Player 1 takes 8 holds: 5 8
4. Player 2 takes 9 holds: 2 9
5. Player 1 takes 4 holds: 5 8 4 stopping Player 2 from taking 4 to win with 2+9+4
6. Player 2 takes 3 holds: 2 9 3 stopping Player 1 from taking 3 to win with 8+4+3
7. Player 1 takes 6 holds: 5 8 4 6 winning anyway with 4+5+6.

Number Scrabble places an enormous cognitive load on the player, who must be continually making trial additions of various combinations of counters: his own,

his opponent's, plus what-if selections from the pool. However, as Newell and Simon show in *Human Problem Solving*, the game is mathematically *isomorphic* to Tic-Tac-Toe – as you'll see if you arrange the nine counters into the magic square SATURN. Accordingly I surreptitiously equipped my 'weaker' student with a Tic-Tac-Toe board marked with the numbers of SATURN, plus instructions to play Tic-Tac-Toe as player 'O', simply calling out the number of the square he placed an 'O' in, and mark his opponent's chosen numbers on the board with an 'X'. Player 'O' gets to start first, and should always choose 5.

Here's the state-of-play in the above game after step 5:

4/O 9/X 2/X

3 5/O 7

8/O 1 6

from which you can see straightaway that 'X' cannot win, because 'O' can select either 3 or 6 to win. Even to spot this fact is a difficult task in Number Scrabble. But not in Tic-Tac-Toe.

What was the purpose of this class demonstration? That talismans really work? Yes, but not in the superstitious sense. The design of any interactive computer system should display its information in a choice of cognitive domain that the user is most proficient in. In the present case, one which demands simple spatial awareness is demonstrably better than one which demands mental arithmetic on simultaneous expressions. You might go so far as to say that, whatever the application, if the programmer can discover a 'magic square' for it, he or she may have the key to enhancing the end-user's productivity quite significantly.

It's obvious when you put it like that. However many programmers think it's best to present the information to be manipulated in a 'traditional' form, i.e. one which might have been used by the operatives before the task was computerised. Taken to its extreme, such programmers would force switching logic designers to work with a display of Aristotelian syllogisms (a mediaeval form of symbolic logic). Perversely, that might be *just* the interface to use if you had to recruit your logic designers from a pool of classics graduates!

Returning to magic squares, might it be that JUPITER (order 4) holds the key to playing an extended Number Scrabble with 16 counters, from which *four* must be selected to add to 34 (34=total 4)? Alas, no.

Let us run fn: comb once more to find all sets of 4 distinct integers taken from 1..16 and adding to 34:

```
      ⍴comb 4
86
```

Now JUPITER exhibits only 10 of these 86 'magic quadruples' (4 rows, 4 columns, 2 diagonals). Which makes me suspect that there are several quite distinct magic squares of order 4, which cannot be transformed into each other by permuting rows, columns, or transposing the matrix.

Is it feasible to generate magic squares systematically? Yes, it's not at all difficult to generate a magic square of any choice of odd order. The fn: gen1 (listed below) does so, but only works for odd arguments. The magic squares it generates are equivalent to the classical ones under row/column permutation. Thus, using the fn: rc (also listed) to permute the output of gen1 we can show that all these APL expressions yield the value 1 (true):

```
SATURN≡gen1 3
MARS≡4 1 3 5 2 rc gen1 5
VENUS≡2 5 1 4 7 3 6 rc gen1 7
MOON≡7 2 6 1 5 9 4 8 3 rc gen1 9
```

The fns listed below form the beginnings of a toolkit to search for magic squares of any order. The new Dyalog feature *dynamic function definition* is used for the first 5 functions (but these can be readily rewritten as conventional 'del' functions):

```
isnatural←{(⍳⍴⍵)≡(,⍵)[⍋,⍵]}
eq←{(,1)≡⍴∪,⍵}
total←{0.5×⍵+⍵*3}
dia←{(((¯1+⍳⊃⍴⍵)⊖⍵)[1;]}
aid←{(((-⍳⊃⍴⍵)⊖⍵)[1;]}

∇ Bool←magic z
[1]    ⍝1 iff z is a magic sq
[2]     Bool←(isnatural z)∧eq(total⊃⍴z),(+/z),(+⌿z),(+/dia z),(+/aid z)

     ∇ poll sec
[1]    ⍝look at ⍴z at: sec intervals
[2]     :Repeat
[3]         '>>> ⍴z=' (⍴z)
[4]         ⎕DL sec
[5]     :Until 0
```

```
     ∇ z←comb order;act;sh
[1]   ⍝ways of combining: (order) nos from: ⍳order*2 to add to: (total order)
[2]    act←act1 ⍝--for different actions on all combinations
[3]    sh←{} ⍝--no trace
[4]   ⍝sh←+ ⍝--to trace the combinations tried
[5]    z←0 ⍝--used as an accumulator by: act
[6]    poll&1
[7]    0(order comb_sel act)⍳order*2
[8]    ⎕TKILL ⎕TNUMS ⍝--terminate: poll

     ∇ jk(order comb_sel act)in;i;j;ijk;me
[1]    ⍝apply act to all combinations or order nos in: in
[2]    me←⍕⍵⊃'order'(⍎⎕SI)'act' ⍝--forms a recursive fn from this op
[3]   ⍝...so you can safely change the op name and it will still recurse!
[4]    j←⍵⊃jk
[5]    :For i :In j↓in ⍝--only bother with i>j in jk
[6]        sh ijk←i,jk ⍝--form new left arg
[7]        :If order>⍴ijk ⍝--then recurse...
[8]            ijk me in
[9]        :Else ⍝--deepest recursion: call the working fn...
[10]           order act1 ijk in
[11]       :EndIf
[12]   :EndFor

     ∇ order act1 xy;x;y
[1]   ⍝perform the chosen action inside: comb
[2]    x y←xy ⍝--x: vec of (order) combinations chosen from vec: y
[3]    :If (total order)=+/x
[4]        sh'>>>'x(y∈x)
[5]        z,←⊂y∈x
[6]    :EndIf

     ∇ z←ros set;n
[1]   ⍝return nos corresp to bitstrings set
[2]    n←⍴⍵⊃set
[3]    z←set/⍨⊂⍳n

     ∇ z←gen1 n;y;i;j
[1]   ⍝generate magic sq: 1st strategy
[2]    j←⌈n÷2 ⍝--mid-pt no
[3]    z←n n⍴⍳n ⍝--shaped array
[4]    z←(j-⍳n)⌽z ⍝--upper latin sq with iss property
[5]    y←⌽z ⍝--reverse it to get lower latin sq
[6]    z←y+n×z-1 ⍝--combine z and y

     ∇ z←p rc m
[1]   ⍝apply perm: p to rows/cols of m
[2]    z←m[p;p]
```

APL Power Shootout: APL+Unix *vs* APL/M *vs* SAX (Part 2)

By Bob Hoekstra (Bob.Hoekstra@khamsin.demon.co.uk)

Abstract

Continuing from last issue's article, we compare the compatibility, performance and features of *APL+Unix version 5.3.2* from APL2000, *Dyalog APL/M version 8.2.2* from Dyadic Systems and *SAX (SHARP APL for UNIX) version 5.0.0* from Soliton.

Introduction

This article is the second of a two-part series, and completes my reviews of Unix APLs (at least for the moment). It follows on from the article in Vector Vol. 16 No. 4[1].

For those who missed the previous instalment, it is available on the Vector web site at www.vector.org.uk/v164/bob164.htm .

More on the Installation

Solaris 8

Since part 1 went to press I have upgraded my Sun Ultra 5[3] workstation with a new 20GB hard disk containing Solaris 8, the latest version of Sun Microsystems's version of the Unix operating system. Naturally I was curious about any performance differences, so immediately installed the APL interpreters under this new OS.

Both APL/M and APL+Unix installed under Solaris 8 exactly as they did under Solaris 7, but I had great difficulty with SAX: I never did get the NSVP processor running and thus could not run SAX under Solaris 8. I suspect the problem may be that Solaris 8 is more critical of daemons running on it. I tried installing SAX while running Solaris 8 in 32-bit mode, but this did not help. I did not spend too much time on this, or I would not have made the deadline for this article.

I completed this article working in Solaris 7. I did some brief comparisons between the two OSes using the other two APLs though, and found no significant differences in performance or any other aspect. I am confident that Soliton will

address this problem soon, and I do not see this as a major problem with their interpreter.

Solaris 2.6 on a Tadpole

I have also recently come into possession of a Tadpole[4] SPARCbook 3GX running Solaris 2.6. All the interpreters seemed to install on this machine satisfactorily, but with a previous generation MicroSPARC processor running at only 110 MHz and a mere 32 MB of memory, performance is less than ideal. I did not use this platform except to confirm that it works.

APL speed comparisons

Tests were created specifically to compare execution speeds and file system access speeds of the three interpreters. Because of other system activity and the state of workspaces, timings were not constant when tests were repeated, but all tests were repeated enough times to ensure that the results were representative.

In all cases, I was very careful to run **identical APL code** on the three interpreters, which meant "standard" APL was used. I also tried to level the playing field by creating approximately the same size of workspace throughout: 4 MB.

Code performance

I spent quite a lot of time timing various bits of standard APL code, but there were few conclusive results. Generally, APL+ seems to be the fastest on most code, with APL/M following close on its heels (differences in time taken are typically less than 10%).

SAX generally came last, but again the difference in expired time (between APL/M and SAX) was only of the order of 10%, and frequently less. SAX seemed to perform worse when there was more looping involved. Some code examples showed a slightly different picture, with SAX coming almost level with APL+, followed closely by APL/M.

The differences shown here are not conclusive. After the comparison was completed I tweaked some of the APL/M code, converting some of the code to dynamic functions. In all cases I got a significant speed-up with dynamic functions: a minimum of 10% and one case (which lent itself to tail recursion) a reduction in processing time of nearly 50%! I am sure that much of the code could have been improved for the other interpreters as well, but I didn't pursue this as deadlines were looming.

Furthermore, Soliton is due to come out with a new version of SAX shortly. I have heard that some of the work that has gone into the Linux version has meant that they have got significant performance improvements in the next generation Solaris version. If this is true, then my results could be back to front by the time a reader of this article makes a purchase.

In any case, I would be very surprised if the differences in performance between these interpreters were significant. By that I mean that these differences would be swamped by the effect of hardware improvements, say adding memory, upgrading the processor speed, or getting an extra processor. Even a new disk drive (say a 10,000 rpm disk replacing a 7,200 rpm disk) could make a difference which would make these performance differences seem irrelevant.

If there is a lesson here, it is that all three of these interpreters perform adequately. If you have to choose which to buy, basing your choice on performance is unlikely to be a wise decision. Let's face it, if performance were that critical, you would probably want to think about compiling at least part of the APL code to maximize performance.

File performance

Here differences were far more marked. I timed (a) the creating, "stuffing" and untying, (b) tying, reading and untying and (c) tying and erasing of component files in a tight loop. To get a reasonable idea, there were two basic file types: "small", where the component contents consisted of the atomic vector, and "large", where the components comprised a 20×20×20 numeric matrix. The loop was such that the product of the number of components and the number of files was 3,000.

In all cases, APL/M was significantly faster than the other two APLs. APL+ came close on the "small" files, but fared less well when the components were larger. SAX was tested with the two extreme synching modes: no synchronising (fsync=0) and "full" synchronising (fsync=2). The latter was very slow indeed, generally taking 2 to 3 times as long as the former, which in turn was significantly slower than the APL+ performance in most cases. The only exception was the tie/read/untie test, were APL+ and SAX (with fsync=0) were more or less level, but both still slower than APL/M.

No doubt a very small part of the poor performance of SAX is that it performs less well in loops, but this cannot possibly be all the difference (or even a significant part of it). Perhaps more interesting is a comment in the manual that, after a file has been created, SAX has this as a shared tie, rather than an exclusive tie. I

suspect this carries a performance penalty. Note that the tie/read/untie cycle compares quite favourably with APL+. But this does not explain the difference in the tie/erase cycle.

An interesting point about APL/M's good performance is that the resulting files seem much smaller. I cannot explain why, but a component file containing 100 "large" components was 1,603,788 bytes in the Dyalog component file, but 3,227,428 bytes for SAX and 3,203,856 bytes with APL+.

This points to a slightly different conclusion than the code performance because the differences in performance are so much greater. For an application where file system performance is critical, APL/M might be a good choice.

Features

Dyalog APL/M

This appears at first sight to be the most feature-rich of the three interpreters. Immediately obvious is the GUI development environment. For the developer this is absolutely wonderful, and once you have grown accustomed to it, it is difficult to do without. However, I did come across a few glitches: editing windows would suddenly (and for no good reason) disappear behind the session manager when I opened a traces window on a variable, and my only crash happened while I had many windows open (I think it was more than 30). Furthermore, the syntax-sensitive colouring doesn't seem to work quite as well as it does in Windows.

The GUI environment stretches into the code as well, with the very easy creation of GUI objects which can be used during run-time. Attractive and intuitive applications can result from this, and perhaps the greatest strength of the APL/M GUI area (which neither of the other interpreters have) is that code for Microsoft Windows is so easily ported to Unix (well, actually Motif). Even this has a few little buggettes though, as the Motif widget sizes are not quite the same as those of Windows, ported applications need a little tweaking to get positioning right. Sometimes the automatic positioning gets it a little wrong as well. But these little problems should not detract from what is a great feature.

Dyadic's APL interpreters are known for their support of namespaces. This is an integral part of APL/M as well, and can help keep code neat, as well as allowing code imports with less fear of name clashes. GUI objects are namespaces in themselves, as are TCP/IP interfaces (the existence of which is a feature in itself).

But the features do not stop here. My personal favourite is dynamic functions, which could be seen as Dyadic Systems's response to J and direct definition all in one. It can give code which is extremely elegant, easy to maintain and lightning fast.

There are several other great features, not least the ability to create user defined operators. Personally, I do not use this much, but one feature that I do use constantly while debugging code is the ability to step through code using the windows that can be set up to appear automatically when code crashes. Even the fact that function editors are separate windows (i.e. the session manager is directly available while you are editing a function) can be a great productivity aid.

APL/M also supports a very full implementation of control structures. This is largely the same as the control structures in APL+.

Then there is the high degree of compatibility between the different versions. APL written for any Microsoft operating system will usually just run. Of course there are things like different naming conventions for files that might be a problem, but the code (including GUI generation) will run. Component files will have to be ported, but Dyadic Systems provide the tools to do this.

Lastly, many people might value the compatibility with other APLs. Using the migration level system variable, the developer can tailor the compatibility to APL2, which sets the standard in many ways.

APL2000's APL+Unix

By comparison, APL+ seems almost devoid of features. However, this is not quite true. No GUI, no syntax colouring. In fact, a bit like the STSC APL*Plus for DOS many years ago.

Of course, this is unfair. While there are few cosmetic features, there are many really useful ones, like a good TCP/IP socket interface. The editor has hardly changed since the early STSC days, but it works well. There are several features which point towards compatibility with APL2, and the language features are slowly being extended, e.g. scalar dyadic functions with axis are supported, as is n-wise reduction and APL2-style function attributes.

There is also support for control structures, more or less identical to those in APL/M. Also, component files have the same structure between Unix, DOS and Windows, meaning that one has data portability between platforms. Note that

this is the opposite of Dyadic Systems's implementation, where the workspace has the same format but not the component files.

The user commands are a very useful feature. One of the many useful sub-features that this creates is the concept of packages. This is somewhat different from the packages of SAM and SAX, but many of the concepts are similar.

To me the most important thing was that, while there were no features that APL/M didn't have, every feature worked perfectly every time. APL+Unix did not crash once. Not even a hiccup.

Soliton's SAX

SAX also has fewer obvious features. But then, SAX is a slightly special case as the APL is somewhat different. It shows a family resemblance to J in many respects, introducing the concepts of rank, frame and cell which the others don't have.

The existence of the NSVP is a very strong plus point. This could be invaluable to those who want to access say a DB2 database on a mainframe from a Unix APL session.

The concept of "intrinsic functions" - essentially an easy interface to compiled code - is also a very strong feature. This can be done in both the other interpreters, but it would appear that the SAX implementation allows the greatest flexibility.

As yet, there is no GUI implementation, but with the recent announcement of the Java interface for SAX/Linux, I suspect that the other Unix versions will get a Java interface as well.

SAX is highly compatible with SAM, the mainframe version. This extends to the use of the same rather clumsy editor. This takes some getting used to, but eventually becomes quite usable (although I am still not fond of it). The other powerful feature that both share is packages, which can be used to squeeze large workspaces into only a small amount of space. This might not be all that useful now that memory prices have decreased, but like APL/M's namespaces, packages can also be used as a name clash preventative.

I suspect that this interpreter will have many enhancements soon and will become a very strong contender indeed. Having said that, it has a language implementation which many will like - it took me a while, but it is definitely growing on me.

Recommendations

It is impossible to recommend purchase of one APL above the other. All of the interpreters reviewed are "good", though not necessarily in the same areas.

If you have used a Dyalog interpreter in the past and liked them, then APL/M is definitely the right choice. You will feel at home immediately. The portability of workspaces from DOS or Windows is also a very strong point. This interpreter might also have some additional benefit if file system performance is critical.

Alternatively, previous users of STSC, Manugistics or APL2000 interpreters would get on much better with APL+Unix. A particular strong point here is compatibility of component files across architectures. This means that one could keep APL running on multiple operating systems with a single data source.

Last, but by no means least, SAX would suit those who want to port from SAM on a mainframe. It might also be better for those who have been using J and don't want to give up all the benefits of this advanced APL derivative.

The only suggestion I can give to the prospective purchaser is to read the features section above carefully and decide which points are applicable in the situation.

Thanks

I thank the manufacturers of the 3 interpreters reviewed. All have helped me at some stage.

Notes, References and Web Sites

1. Refer to *APL Power Shootout, Part 1*, Vector Vol. 16 No. 4, *My SAX Experience*, Vector Vol. 16 No. 3, and *Dyalog APL for Motif Version 8.1 Release 2 for Sun Solaris 5.5*, Vector Vol. 14 No. 3, all by Bob Hoekstra.

2. The web site for Soliton is http://www.soliton.com, APL2000 have a web site at http://www.apl2000.com/ and Dyadic Systems's web site is at http://www.dyadic.com/.

3. The web site for Sun's Ultra 5 page is http://www.sun.com/desktop/products/ultra5/ .

4. Tadpole and RDI now share their web prescence at http://www.rdi.com/. They were competitors in the portable SPARC market, but joined forces some time ago. Unfortunately my SPARCbook 3GX (being an older machine) gets no mention on the site.

Introduction to Tree Searching in APL: Using the Tic Tac Toe Game in 3D

by Dan Baronet (danb@dsuper.net)

Introduction

This article deals with tree searching using a popular game as medium. The code described here can be used in many situations to traverse a "tree" of possibilities.

Keywords

This document discusses computer science subjects including *recursivity, tree searching, heuristic evaluation* and *sequential machines.*

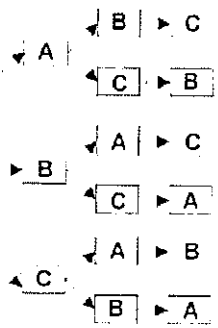
Examples

Tree searching is a technique that can be used to solve large-scale problems. Many situations can use tree searching, some more interesting than others. Possible examples are permutations, where all possible combinations of a set are being produced, and calling tree analysis, where all identifiers of a program are enumerated. This technique is used effectively in situations where others, even if possible, may not be practical, typically because they involve a large number of possibilities.

Notions and Terminology

To better understand the concept of tree searching, consider the permutation case. If we were to show all the possible permutations of the letters A, B and C we would get 6 cases. The result would essentially consist of each letter followed by the permutation of the remaining letters in a "tree like" fashion, in a repetitive, recursive process.

Schematically it could look like the picture below, where we first generate all possible letters, then the remaining letters and, finally, the last letter.



This picture represents the whole *tree*. Each box is a *node*. Each combination of arrow/box is a *branch*. The point where all branches originate is the *root* and where they end is a *leaf*. The length of a branch from the trunk is its *depth*.

Not all trees are shaped like this one. Like biological trees, most have branches of various depths. Unlike biological trees, they don't grow while you search them because they are static, at least the kind we will be dealing with.

Assuming that no setup is necessary, searching a path consists of two steps:

1. check for a solution and **generate** new branches
2. search each one of the new generated branches

This is a recursive process.

Permutations

The previous example can be solved many (and even better) ways but we'll use the tree technique as this is a good example.

Here we know we have a solution when no more branches may be generated. So,

1. check for solutions and generate new branches

The following code will do just that.

```
    ∇ unused←check_generate used
[1]   unused←'ABC'~used ⍝ return letters not in the branch
[2]   :if 0=⍴unused  ⍝ are we done?
[3]   ⎕←used ⍝ show the result
[4]   :endif
    ∇
```

This code merely consists of generating the letters NOT used in the branch. If the branch we are dealing with is a solution we display it.

This is it!

2. search each one

```
    ∇ trybranch branch; new; br
[1]  new←check_generate branch ⍝ the new material
[2]  :for br :in ⍳⍴new ⍝ try them all!
[3]  trybranch branch,new[br]
[4]  :endfor
    ∇
```

As we see, the function calls itself for each new branch generated.

If we run this code *as is*, all permutations of the 3 letters 'ABC' will be displayed. To display other permutations the function must be modified or a global variable used to hold the set to permute.

Many other problems can be solved with this technique. There is the 8 queens problem, the knight's tour, the "knapsack" problem where, given a layout of pieces and rules for moving them, we must find the moves to put them in another configuration (Rubik's cube could also fall in this category).

A more practical use is to generate the calling tree of programs, where all of the elements needed to run a program are identified.

All the above require their own *generation* function.

Let's have a quick look at the 8 queens problem.

The 8 Queens Problem

This problem is often assigned to computer science students to solve.

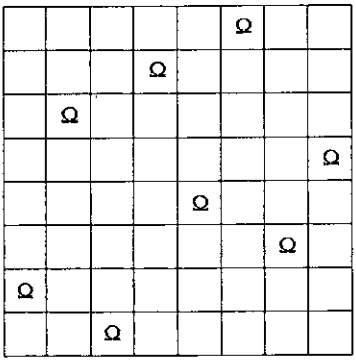
The idea is to put 8 queens[1] on a chess board in such a way that none can capture another. There are 8!64 ways to put them on the board (4,426,165,368) but far fewer unique solutions[2]! Trial and error is an unlikely recipe for success. A possible solution is shown on the next page.

We could, of course, generate all combinations and screen out unwanted ones but with today's machines it would take a fair amount of time (not to mention space if done all at once, the APL way). This is another good example for using tree searching techniques.

---

[1] In a chess game the queen may move in all directions: horizontally, vertically and diagonally.

[2] Exactly 37 *unique* solutions.



A method

We put one queen down, note all the places where another queen can be placed without being captured, try the second queen on one of these, note the remaining positions, etc., continuing in this manner recursively.

Even with this method we're not guaranteed a timely success. There are 64 possible squares for the first queen, 42 for the 2nd, 30 for the 3rd, etc. This can rapidly get out of hand. What we need is fast discriminatory technique.

What do we know

One thing we know is that each queen will appear only once per row AND column. We can reduce the number of searches like this:

- try the 1st queen on each square of the 1st row
- try the 2nd queen on each square of the 2nd row (except where the 1st queen can capture it)
- try the 3rd queen on each square of the 3rd row (except where the 2 other queens can capture it)
- and so on

We know we have a solution when the branch is 8 steps long (when all the queens have been successfully placed on the board).

The generation function could look like this:

```
    ∇new←search_generate br; n ⍝ ⎕io 1
[1] new←(⍳8)~br,(br+n),br-n←⌽⍳⍴br ⍝ All safe squares to try
[2] :if 8=⍴br ⍝ Do we have a solution?
[3]   ⎕←br    ⍝ Display the solution found
[4] :end
    ∇
```

When to stop searching

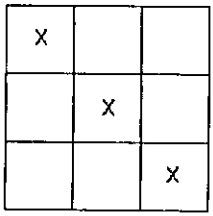
This is an important point. Do we want to search the ENTIRE tree or do we prefer to find only ONE solution? In the second case we must signal the end of the

search and code for it. Naked branch is only good if no result must be returned. A better way to do this is either to `⎕signal` and trap or set a global variable and use it in the **trybranch** function as in

```
:if SolutionFound :return :end
```
[3]

An even more interesting example: Tic Tac Toe

Overview



Everybody should be familiar with the 2-dimensional version of this game. This is the game of 9 squares (3 rows and 3 columns) where two players try, in turn, to acquire 3 squares (slots) lined up in a row. Like on this picture, to the left.

All positions (slots) a player acquires are marked with a symbol different from the other player. Typically, X and O are used as symbols.

The version we'll be using is bit more elaborate but with the same rules.

Model

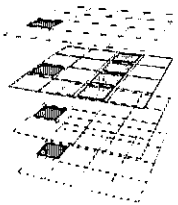
The game discussed here is a 3-dimensional model (a *cube*) where 2 players try to line up 4 squares in any spatial direction.

The *cube* consists of 64 positions, or *slots*, of 4 *planes* of 4 rows by 4 columns where players make their moves alternatively, marking a new *slot* of their choice each time.

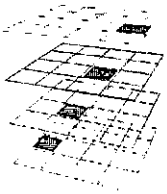
The *state* of the game changes from move to move. A typical game starts from an initial *state* where typically none of the *slots* are used to a final *state* where one of the players has acquired 4-lined up *slots* and the game is over.

[3] hardcore APLers may wish to use the expression '`→SolutionFound/0`' instead.

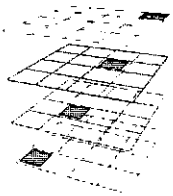
Examples of possible lineups are:



In this example 4 *slots* have been lined up vertically and 4 others horizontally.



Here the *slots* have been lined up "sideways".



Here the *slots* have been lined up from one top corner to the opposite bottom corner.

There are 76 different possible ways to line up 4 *slots* in this cube.

Each *slot* may be part of 4 or 7 of these *lines*.

There are 18 different *planes*[4] and each *line* may be part of 2 or 3 of them.

'X' and 'O' to represent the 2 players.

Discussion

In computer terms a typical game consists of

- Initialize the game
- Repeat
    1. your turn
    2. my turn
    3. display the state/cube
    4. Until the game is over

The game interface is of little interest here. It can be line by line, or with graphics. Graphics may be Windows-style or even *openGL* (see the various implementations listed below)[5].

[4] 4 on the Z axis (those we see in the pictures), 4 on the Y axis, 4 on the X axis and 6 more tilted.

The section of interest for us is where we attempt to determine, from the state of the game, if a winning solution is possible. This is where tree searching comes in handy.

Implementation

It is possible to detect a winning state by simulating moves. For example consider a single[6] *plane* of the cube like the one to the right where each *slot* has been attributed a name (number) and the 'O's occupy positions 41, 43 and 23.

11	12	13	14
21	22	**23**	24
31	32	33	34
41	42	**43**	44

If it is 'O's turn to move then it can win within 6 moves by playing, for example, 14, 33, 44, 24, 22 and either 11 or 21.

The other player's moves are predictable. S/he will be forced to play, at each corresponding move, to 32, 13, 42, 34 and either 21 or 11.

We need a function such that

Winningmoves_ FindWinningMoves gamestate

will find a winning solution.

Or, since *gamestate* is a function of X and O we can use:

Winningmoves_ X FindWinningMoves O

Winningmoves contains all the moves to play in the order they were found in order to produce a winning situation. Its format may also include the opponent's move for practical reasons.

With such a function defined the bulk of the work will consist of building an interface around it. That part will not be covered here.

---

[5] For those who cannot wait go to www.dyadic.com's web server page or directly to 195.212.12.1:8081/ttt0.htm for an example of an interface.

[6] There are many other planes in the cube. For the sake of this example the other cube slots are not taken in account.

Details

The state representation

We must be able to represent the *state* of the game in a practical manner. Marking the 64 positions with X and O will uniquely represent the *state* of the game but is of little use.

Since we're looking for complete *lines* a better representation of the *state* of the cube would be to code the number of X and Os on each *line*. This will require 76 integers (one per possible winning alignment) but will provide us with more valuable information, namely whether we (or the opponent) have lined up 4 *slots* or not.

For example a *line* with 4 Os could have the value 4 and a *line* with 4 Xs the value 40. A *line* with 1 X and 3 OS the value 13 and so on, i.e. exactly the number of Os on the *line* + 10 times the number of Xs[7].

Initially, since the 76 *lines* have no X or Os in them, we have a *state* of 76 zeroes. At the end there will at least one 4 or one 40 in them but not both.

The tree searching function

This one is easy. All we need to do is generate new branches and run them through the same function. Like the **trybranch** function above. This time it must deal with matrix results:

```
    ∇ Runbranch node; newbranches; br
[1] ⍝ Get <Generate> to provide the new branches for this node
[2] newbranches← Generate node
[3] :for br :in ⍳1↓⍴newbranches ⍝ For each newbranch
[4]   Runbranch node,newbranches[;br] ⍝ Try it
[5]   →SearchOver/0 ⍝ Are we done? Then destack.
[6] :endfor
    ∇
```

The function must also be able to stop searching when we decide so. After all, we're only interested in ONE solution. Line [5] takes care of this.

The *new branches* generating function

This function is responsible for generating all possible branches.

---

[7] The value 10 is for visual appearance only. In fact, any value above 4 would do.

The idea is to generate moves which continually line up 3 *slots* and effectively *force* the opponent until s/he has to play at more than one place at once. We then win at the next move.

We start in a specific *state* with Xs and Os laid out. The two variables 'FREE0' and 'STATE0' contain information related to this initial *state*. These are used to create their local varying counterparts 'Free' and 'State' at each level of the search.

After any series of moves, if 'State' contains 30 we know we can win right away. If 'State' contains 20 we know we can further try as we have 2 *slots* on at least one *line* and we can force our opponent to play on each one of the remaining pair of *slots* on each one of these *lines*.

There is also a special case where we might be forced ourselves to play but which is of no consequence if our response also further forces the opponent to play.

Here's the code to do the above. `ALL` is all the 64 *slots*. `LINES` is the 76x4 table of *slots* for each *line*.

```
     ∇new←Generate xo;State;Free;sl;li
[1]  new←0/xo ⍝ assume no possible new move
[2]  Free←FREE0∧~ALL∈xo ⍝ free slots=original free minus those used so far
[3]  State←STATE0+stateof xo ⍝ state of each line=original plus branch state
[4]  :if 30 ∈ State ⍝ Can I win? (3Xs, no Os on a line)
[5]  Sequence←xo,2↑frees sline 30 ⍝ Yes, take note of the winning sequence
[6]  SearchOver←1      ⍝ End the search
[7]  :elseif 3 ∈ State ⍝ Can s/he win? If so WE are forced to play.
[8]   :if 1=⍴sl←frees sline 3 ⍝ If at more than 1 place then
        this is hopeless
[9]    :if ∨/li←(State=20)∧∨/LINES=sl ⍝ Can we also force opponent
        when we play there?
[10] new←2 1⍴ (sl≠1↑li) ⌽li←frees,li≠LINES⍝ Indeed. Find where s/he
        is then forced.
[11]   :end      ⍝ And keep'on searchin'
[12]  :end       ⍝ We were in a fork 8- forget this branch
[13] :elseif 20 ∈ State      ⍝ we cannot win but we might be able
        to further force opponent to play
[14] new←State BEST frees sline 20    ⍝ Order all slots where
        we force opponent
[15] :end ⍝ no more possible move
     ∇
```

`<sline>` returns the ravel of the *lines* whose *state* is the same as its argument. `'sline 30'` returns the *lines* with state 30.

[8] A fork is when one must play at several places at once in order NOT to lose.

<*frees*> returns the *slots* which are free in a list. Thus '*frees sline* 30' returns the unused *slots* in the *lines* X which have 3 *slots* lined up.

<*stateof*> returns the state of Xs versus Os as defined previously.

How to find a solution fast

We must find a way to get rapidly at the solution otherwise we could spend a very long time traversing the tree. One way is to attribute a *weight* to each position. To do that we can first attribute *weights* to lines, and for each slot sum up the values of each line it is used in. For example, if position P1 is member of 3 lines whose values are 10, 0 and 20 then P1's *weight* is 30. If another position's *weight* is more than P1, it will be tried first.

For example, in the following picture, X can force O at 7 places, only one of which will ensure a quick win. All the others may lead to a winning situation but only position 34 can create a fork and will do so in 2 moves.

11	12	13	14
21	22	23	24
31	32	33	34
41	42	43	44

Each slot is traversed by 2 or 3 lines here. Slot 11 is in lines 11-14, 11-41 and 11-44. But playing there is obviously not optimal.

On the other hand, slot 34, which is only in lines 31-34 and 14-44 is the best choice, because lines where X already has 2 tokens lined up are more valuable than those where X only has 1 token. By choosing the right *weights* for each *state* we can achieve the desired result. These weights must be guessed or evaluated heuristically. In the following code, the function <*BEST*> finds the best slot this way.

Making the best choice

The <*BEST*> function orders the slots to reduce searching. It puts the "best" slots first according to the state of the lines. There are many ways to do this. The method used here is simple.

It first attributes a value to each *line* according to its state. Here a *line* gets the value 1 if it has 2 Xs on it, 0 otherwise. It then attributes a value to each *slot* by summing the value of all the *lines* it belongs to (a *slot* may belong to 4 or 7 *lines* in this *cube*, information held by variable *ISL*).

Thus:

```
    ∇sl←State BEST slots; value; ord
[1] value←0,State=20 ⍝ value of each line
[2] value←+/value[ISL[slots;]]
[3] ord←⍒value ⍝ which ones first
[4] sl←slots[ord,[-.1]ord+¯1*ord]      ⍝ alternate pairs
    ∇
```

Even with this method the tree it generates can be enormous. We must be able to "prune" it by specifying a minimum value for each *slot* and/or a maximum number of branches. Both can be specified at each level of the tree. To keep things simple we'll use a maximum value of 4 such that line [3] now becomes:

```
[3] ord←(4⌊⍴ord)↑ord←⍒value
```

Putting it all together

We now need a cover function for this. We need to initialize the *STATE0*, *FREE0* and the *SearchOver* variables, as follows:

```
    ∇ Sequence←X FindWinningMoves 0;FREE0;STATE0;SearchOver
[1] ⍝ Find a winning sequence for X over 0
[2]  STATE0←(+/LINES∈0)+ 10×+/LINES∈X
[3]  FREE0←~ALL∈X,0
[4]  SearchOver←0
[5]  Runbranch Sequence←2 0⍴0
    ∇
```

Sequence, the result variable, is set prior to calling <*RunBranch*> just in case no solution is found. In the event that we DO find a solution, <*Generate*> will take care of setting it properly.

This is where the ability to see and reset localized variables, on the stack, can be useful. In J, for example, this would not be possible. Instead the solution can be temporarily stored elsewhere, in a locale for example.

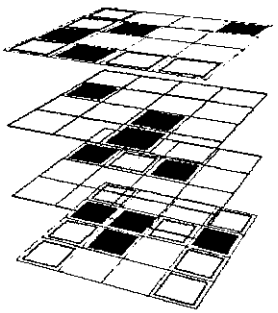
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Let's try it on a simple case:

```
0 1 11 15 FindWinningMoves ''
3 7
2 0
```

Meaning "I will play 3, O will play 2, I will play 7 (and win)".

OK. Let's try something more serious. In the following picture we have



```
X←15 9 24 18 57 40 42 22 1 56 4 59 53

O←12 3 48 51 60  0 63 41 2 52 8 55 58
```

(cell 0 is the top leftmost one).

To see if X can win we do:

```
    X FindWinningMoves O
 5 37 26 6 38 25 32 36 39
13 21 30 7 54 27 47 44  0
```

and X is assured victory in 9 moves *or less*. BUT! If it's O's turn to play s/he can win in 3 moves:

```
    O FindWinningMoves X
38 35 44
25 19  0
```

Improvements

There are many ways to improve the search. We can

- fine-tune the pruning algorithm with better weights and functions
- allow the program to find a shorter solution
- specify a maximum search level to limit *"intelligence"*

To name a few possibilities.

In the various implementations below, some of these improvements have been made. Among others, the "BEST" algorithm uses values that have been reached after trying several evaluation methods (hence the term *heuristic evaluation*).

Conclusion

There are many ways to tackle problems.

This tree searching technique is very good at going through large amounts of cases or situations.

Like every other technique one must exercise judgment when using it. If used without preparation it can be slow to produce results, if at all.

Various Implementations

My first attempt at writing this game goes back over 20 years! The first version was line by line, with quote-quad for input. A similar version exists for SHARP APL, APL*PC and J.

After graphics became popular I ported the game to APLPLUS*II then Dyalog APL. A J version now also exists using graphics.

All the code can be found at www.dsuper.net/~danb/milinta.

ZarkWin: Windows Programming without ⎕WI

Gary A. Bergquist

This paper was presented at the APL2000 Users' Meeting, Orlando, Nov. 1999

Summary: ZarkWin is an application development tool developed in APL+Win for building Windows-style user interfaces. The mission of ZarkWin is to enable you to build user interfaces as quickly as you can design them. The author believes that a disproportionate amount of time is being devoted to implementing fancy user interfaces. ZarkWin shortcuts the implementation process by fighting fire with fire. A robust full-screen interface is used to build robust full-screen user interfaces. Naturally, ZarkWin was built with ZarkWin.

The Concept

Experienced APL programmers tend to take for granted the magic of APL. Sure, they appreciate the power and productivity of APL, and enjoy working with it, but they don't often stop to wonder *why* it works so nicely. Is it because of its mathematical orientation? Its symbolic nature? Its array-handling? Its immediate execution mode? Its obsessive consistency?

Yes to all of the above. But, more generally, what gives APL its glow is its similarity to a straight line: it *is* the shortest distance between two points. The two points, of course, are the start and end of the job. APL is the shortest distance from the statement of the problem to the implementation of the solution.

This is no accident of nature. Rather it is APL's simple adherence to the laws of nature. In particular, APL abides by what we might call nature's Law of Abstraction:

> *If you can devise a reasonable abstraction of reality (vectors, matrices, nested arrays) and of the needs of the real world (reductions, selections, sorting, relations, inner products), you can design a mechanism (symbols, arguments, results, operators, defined functions, immediate execution mode) that allows you to solve real world problems rapidly because of the direct translation from reality to the mechanism, by way of abstraction.*

In other words, the more closely the mechanism reflects the characteristics of the problem, the more rapidly you can solve the problem.

The Reality

Frankly, I miss the days when the user interface was implemented in APL like this:

```
[1]   'Enter name:'
[2]   NAME←⍞
[3]   'Enter age:'
[4]   AGE←⍞
```

There was no excess baggage. You could implement the "user interface" as quickly as you could imagine it.

No longer.

Now we talk about classes, properties, styles, events, and so on. We converse in a jargon that has nothing whatsoever to do with NAME and AGE. We become intimately familiar with a system function named ⎕*WI*, which has a vast syntax and which is pronounced "Quad-we" (as in, "We submit") but which should be pronounced "Quad-why" (as in, "Why me?"). Clearly, the line from problem to solution is not a straight one.

Don't get me wrong. I don't think ⎕*WI* should be abolished. On the contrary, since we live in a world of Windows, and since ⎕*WI* is our doorway into the jungle of Windows functionality, such a mechanism is a necessity. It's just not a necessity I want to look at every day. It's too tedious. Using ⎕*WI* slows me down and detours me from the straight line.

Instead, we should let ⎕*WI* be the foundation upon which our straight-line abstraction is built. Just as APL is built in assembler, or C, or whatever other language is currently appropriate for implementing APL in the real world, let our tools be built in ⎕*WI*. But let's begin the design of these tools by evaluating the user interface at a higher level, from the point of view of what we want it to do, not from the point of view of the system-level functions available to do it.

The Plan

Compared to the simple interfaces of yesteryear, such as the quote-quad and quad code above, the user interfaces of today are quite complex. They look so much nicer and do so much more. This is great for the user but is a major headache for the programmer.

The concept of defining an interface via a function editor is inadequate. With a function editor, you take these steps:

1. Imagine how the form should look.
2. Translate that picture in your mind to the needed commands, and type them into the function editor.
3. Run the application, or at least the interface part of it, to see whether the commands are doing what you want.
4. If it's not quite right, return to step 1 and repeat.

While these steps get the job done, they are too round-about. A more direct approach is to use a visual form editor instead:

1. Imagine how the form should look compared to the form you are looking at on the screen.
2. Use your mouse to drag, click, or double-click the form into shape.

The Design

If it is possible to edit a form, as described above, it is possible to define it. By "define," I don't mean define it in terms of the □*WI* commands required to build it. I mean define it as an array of parameters that have nothing to do with *how* it will be built. For example, here are some form parameters:

- The caption on the title bar, if any;
- The colour of the form;
- The font used on the form;
- Whether or not the form can be resized by the user and, if so, how the objects on the form will respond to the resizing;
- Whether the form will have menu items and, if so, what their captions and shortcut keys will be and what they will do if clicked;
- What objects, such as buttons, edit fields, labels, check boxes, options, lists, and so on, will appear on the form, and where, and how they will behave.

If you are familiar with □*WI*, you are probably contemplating the syntax you would use to set each of these parameters.

Please don't.

Instead, imagine yourself sitting with a user as he describes the desired form. With a piece of paper in front of you, make a sketch of the form and annotate it with all the necessary details. Once you have the form's complete specifications in front of you, design a nested array that contains all of these specifications. For example, the first item can be the form's title bar caption; the second item can be the form's colour, or empty if the default colour is to be used; the third item can be a nested list of the parameters that describe each of the objects on the form, one item per object, where you define the object's parameters any way you want; and so on.

In the process of designing this nested array, try to keep the structure general so you can use the same design for other forms. As you do this, you will find yourself making assumptions and establishing restrictions so you can keep the design simple. By simplifying, you will be limiting the capabilities of forms that can be defined within the confines of this nested structure. Nevertheless, you will likely be surprised at how few those restrictions are and how easily they can be eliminated by tweaking the design of your nested array.

Once such a nested array is documented, it is possible to write a function (using ⎕*WI* of course) that constructs and presents the form defined by that array. Here's how it would work:

```
zwShow formDefn
```

The argument of the *zwShow* function is the nested array that contains everything there is to know about the form. *zwShow* does the tedious work for you. It figures out what ⎕*WI* calls are needed to build the form and make it respond appropriately to the various possible user actions.

Unfortunately, even with such a function, the task that remains is formidable. You must construct the nested array. Since the definition of such a nested array is non-trivial, building or modifying it will be difficult, time-consuming, and fraught with mistakes. However, it is possible to write a function that presents a set of forms that allow you to interactively modify the contents of the nested array:

```
zwDef 'formDefn'
```

The argument of the *zwDef* function is the name of the nested array. The argument is provided as a name rather than as the nested array itself so that *zwDef* can reassign the named variable. The aim of *zwDef* is to present the form as it is currently defined in its named argument, and to allow you to use the

mouse to move form items or to trigger pop-up forms that enable you to add, delete, or modify form items.

This, then, is the design principal of ZarkWin: Design and document a nested array that can contain the definition of any conceivable (reasonable) form. From that documentation, write a function (*zwShow*) that will build and present the form, and a function (*zwDef*) that will allow you to easily modify the nested array. Once *zwShow* and *zwDef* are written, you never again need to get sucked into the intricacies of ⎕*WI*. Furthermore, since forms can be held in your hand as independent nested arrays, you can easily manage them, copying them into the workspace, erasing them from the workspace, or saving each one as a file component.

The Specifics

Up to this point in the description of the ZarkWin approach, the comments have been kept general. When the Zark folks sat down to discuss the design of the nested array form definition, specific assumptions and decisions were made. The resulting design reflects the mental model of what a form is to us. The remainder of this section describes the salient features of that model. The details of the nested array definition are too lengthy to be included as part of this paper.

Control Objects

A form consists of a set of rectangular user controls, called "Control Objects." These objects are defined for the convenience of the ZarkWin user, and don't necessarily correspond to APL+Win control objects. For example, a set of six options (radio buttons) and their captions can be viewed as a single Control Object in ZarkWin, though they are six control objects in APL+Win. A group of three buttons (perhaps labelled OK, Cancel, and Help) can be a single Control Object in ZarkWin, though they are three control objects in APL+Win. A set of three edit fields and their labels (say, Name, Age, and Salary) can be a single Control Object in ZarkWin though they are six control objects in APL+Win (three edit fields and three labels).

Values and Reference Names

Each Control Object is generally defined to have a "value" and is given a reference name to refer to that value. The reference names are like APL variable names. For example, a Control Object of six options is defined to have a value that is the scalar index (origin 1) of the selected option. The reference name might be, say, DEPT or MARITAL. A Control Object of six check boxes is defined to have a

value that is a six-element bit vector, flagging the boxes that are checked. The reference name might be CHKS or BITS. Here again, the ZarkWin meaning of "value" doesn't necessarily correspond to the APL+Win meaning. This is an advantage, since you don't have to remember whether you want the "text" property, the "caption" property, or the "value" property, as you do when working with `⎕WI`.

Subsets

A Control Object can have "subsets." For example, suppose the Control Object whose reference name is EMP contains three edit fields labelled, respectively, Name, Age, and Salary. The "value" of EMP is a three-item nest (e.g. 'John Smith' 35 25000). You may provide reference names for each of the three edit fields (e.g. NAME, AGE, and SALARY), so you can refer to the value of each field individually. In this event, the fields are called "subsets" of the Control Object. If you do not provide subset names, you can still refer to the subsets one at a time by using "pick notation" (e.g. `2⊃EMP` for AGE).

Datatype Conversion and Field Validation

When you refer to the value of a Control Object or one of its subsets, you can expect it to be in whatever form is natural for that Control Object. For example, the value of the AGE field will be a numeric scalar, even though the value is presented in what you know to be an APL+Win Edit field whose 'text' property is the character vector representation of the age. Likewise, the value of a date field will be an integer scalar in the form yyyymmdd (e.g. 19981103). Datatype conversions and field validations are handled automatically by the Control Objects. If the user tries to enter a non-numeric age or an invalid date, he sees an error message appear and is required to provide valid input. You do not have to do any programming to enable this datatype validation.

You do, however, have to provide validity checks that go beyond datatype validation. For example, if the employee's salary must be under $100,000 when the employee is single and male, you must say so within the form's definition. In ZarkWin parlance, you would say something like this:

```
(SALARY≥100000)∧(MSTATUS=1)∧SEX=1 ⍝ Single males can't earn
more than $100,000
```

Other than such application-specific validations, ZarkWin handles the field validation process on its own, without programming or further specification by you.

In general, the field validation process does not take place at each keystroke or when tabbing to the next field. Rather, ZarkWin waits until the form is closed.

Closing the Form

There are two ways to close a form: the OK-close and the Cancel-close. In the former (OK-close), all field validation checks take place. If any checks fail, an error message displays, the focus moves to that field, and the form remains open. In the latter (Cancel-close), the form is immediately closed, whether or not the values are valid. The closing of a form can be triggered by a user (e.g. closing the form by clicking on the system menu), by user action on a Control Object (e.g. clicking on the button labelled "OK"), or under program control, given any condition the programmer deems worthy of closing the form.

Event Handlers

While the aim of ZarkWin is to minimize the amount of APL programming required to implement the form, you may still include any amount of special code by providing expressions that are executed in response to various events. For example, you can provide an expression that is run after the form is constructed and just before it becomes visible. You can use this expression for any last-minute modifications to the form, such as disabling (greying out) certain fields based on the values in other fields, or for resetting lists of available choices, and so on. Another expression is run after the form is OK-closed and the fields have been validated, but just before the form is removed from the screen. You can use this expression to read the data from the form and file it or to do other processing. The result of the expression tells ZarkWin whether or not to go ahead and close the form. Likewise, you can provide expressions to be executed when buttons, or options, or checks, or list items, or menu items are clicked or double-clicked or changed, and so on. In general, the result of the expression should be 1 to have ZarkWin OK-close the form, or -1 to have ZarkWin Cancel-close the form, or 0 to leave the form open.

Program Control of Control Objects

There are three common ways to pass values into the form, and to retrieve them back from the form.

1. Via global variables:

```
AGE←35 ⋄ NAME←'Smith'
R←zwShow formDEFN
```

If the left argument of zwShow is omitted, zwShow uses the values of all existing global variables whose names match those of Control Objects or their named subsets. The global values are used to initialize the values of these objects on the form. Conversely, when the form is OK-closed, the form assigns the values of its Control Objects to the like-named APL variables. The result of zwShow is 0 if the form is Cancel-closed, or 1 if it is OK-closed.

2. Within expressions executed at certain events, while the form is open:

```
'AGE NAME' ∆zw 'value' (35 'Smith')
(AGE NAME)←'AGE NAME' ∆zw 'value'
```

The first expression writes the values given in the second item of the right argument of ∆zw to the Control Objects or subsets named in the left argument. This expression could be included in the expression that is run just before the form displays. The second expression reads from the form and explicitly returns the values of the Control Objects or subsets named in the left argument. This expression could be included in the expression that is run just before the form is removed from the screen.

The ∆zw function can also be used for setting or retrieving or otherwise controlling other properties of Control Objects. Its left argument is always a list (vector, matrix, or nest) of the reference names of the Control Objects or subsets being manipulated. Here are some examples:

Grey-out the second and third subsets of the CHKS Control Object:

```
'2⊃CHKS' '3⊃CHKS' ∆zw 'enabled' 0
```

Set the choices for the COLORS Control Object to Red, Green, and Blue:

```
'COLORS' ∆zw 'list' ('Red' 'Green' 'Blue')
```

Make the SALARY Control Object invisible; make the DEPT Control Object visible:

```
'SALARY DEPT' ∆zw 'visible' (0 1)
```

3. Via arguments and results:

```
N←'AGE NAME'
R←(N ('value'(35 'Smith'))) N zwShow formDEFN
```

When the left argument of zwShow is provided, it has two items. The first item is a nest with an even number of items. The first item of each pair is a suitable left argument to `∆zw`; the second item is the corresponding right argument. These pairs are passed to `∆zw` by zwShow prior to displaying the form. When the form is OK-closed, the zwShow function explicitly returns the values of the Control Objects named in the second item of its left argument. The result of zwShow is the scalar 0 if the form is Cancel-closed, or is a vector of Control Object values if it is OK-closed.

The result of zwShow is empty (`0 0⍴0`) if the form is presented non-modally, i.e. if processing continues without waiting for user interaction.

ZarkWin Implementation

The behaviour of each ZarkWin Control Object is defined entirely by a single pre-defined APL function. For example, the function `zw∆Buttons` defines how sets of Buttons will behave; `zw∆Checks` defines how groups of check boxes will behave. These functions are written to respond appropriately to a variety of possible arguments. Such functions always begin with `zw∆` (e.g. `zw∆List`, `zw∆Options`, `zw∆Form`, ...). If certain Control Objects are not needed in a particular application, the corresponding functions may be erased from the workspace. Likewise, new Control Object functionality comes from simply copying in the new `zw∆___` functions.

Parents and Children

Some Control Objects can contain other Control Objects. For example, the Control Object known as a Selector can contain Control Objects known as Pages. Likewise, Pages can contain any Control Objects except Pages, including Selectors. A Control Object that can contain other Control Objects is called a parent. The Control Objects contained within a parent are called children. A form is always considered a parent.

In general, children inherit certain traits from their parents. For example, the colour and font style used by the text on a Control Object is the same as that of its parent unless you specify otherwise. Likewise, if a parent is disabled (greyed out) or made invisible, all of its children are disabled or invisible. If an input field on a Control Object is modified, the parent of the Control Object is also considered modified.

Parent Control Objects have reference names, just as their children do. The "value" of a parent is a nest of the values of its children. As such, you may set or retrieve all of the values of the children Control Objects of a parent by referring to the parent's reference name.

Groups

One of the most interesting of the parent Control Objects is the Group. A Group is defined as a set of Control Objects that are each positioned relative to one another. For example, suppose we have a Group that contains three Control Objects. The first is a large-font bold label; the second is a set of three one-line edit fields centred below the label and separated from it by two lines; and the third is a set of two buttons centred below the one-line edit fields and separated from them by three lines. The Group derives its size from the sizes and relative positions of its children.

What makes Groups interesting and powerful is that they remove much of the burden of placing Control Objects on the form. For example, suppose you decide to decrease the size of the big bold label mentioned above. After doing so, you will probably have to move the other two objects to maintain the aesthetics of the form. However, if all three objects are "glued" together as children of a Group, when one object changes size or shape, the others quickly hop into place to maintain their relative positioning (e.g. "centred and below by two lines").

Groups can contain any Control Objects as their children except Pages (since only Selectors can have Pages), including other Groups or Selectors.

Because of a Group's ability to keep its children automatically positioned to look good, it is not uncommon for a Form or Page to contain all its Control Objects in a single Group. Once that Group is defined, it is then natural to want the Form or Page to derive its size from the size of the Group, being just large enough to contain its children. When this is the case, you can take a shortcut by simply declaring the Form or Page to be an "implicitly grouped" Form or Page. Then, you don't have to define an explicit Group. Instead, you place the Control Objects directly on the Form or Page and define their relative positioning. The size of the Form or Page is automatically derived from its children.

Resizing

By specifying the type of border a form has, you can determine whether or not the user can "resize" the form by dragging its borders. Forms that can be resized may have any one of the following "resize behaviours:"

- *Remain* – The Control Objects on the form stay the same size and remain at the same location (from the upper left hand corner of the form).
- *Stretch* – The Control Objects, including any text, grow or shrink in height and width by the same proportions the form grows or shrinks. This can result in short-fat or tall-skinny objects.

- *Grow* – The form maintains the original proportion of its height to its width. For example, if the user makes the form 50% wider and 30% taller, the width of the form snaps back to just 30% wider. The Control Objects, including any text, grow or shrink by the same proportion in both directions.

- *Keep Centred* – The Control Objects remain the same size and the same relative distance from one another. As the form grows or shrinks, the extra space is added or subtracted from both edges of the form equally.

- *Realign* – The Control Objects realign themselves independently in any way they see fit. For example, the Buttons might remain the same size and distance from the bottom right edge of the form, while the List, Tree, or Multi-line Edit objects might grow to allow more lines/columns to display. The text sizes do not change, in general.

Miscellaneous

ZarkWin functionality tends to expand as APL+Win provides additional Windows functionality and as users demand more from their user interfaces. Here are some additional ZarkWin capabilities:

- **Status Bar support.** A status bar can be requested at the bottom of a Form. The status bar may be defined to include CapsLock, NumLock, and/or ScrollLock panes, which are placed to the far right and which use the text 'Caps', 'Num' and 'Scroll'. In addition to these panes, you can define the status bar to contain any number of standard panes, one of which can be designated to display a different message when the focus is located in each Control Object. In addition to CapsLock, NumLock, ScrollLock, and standard panes, you can enable an "overlay" pane which (temporarily) replaces the entire status bar and which you can use, for example, to display messages that tell the user the status of lengthy processing.

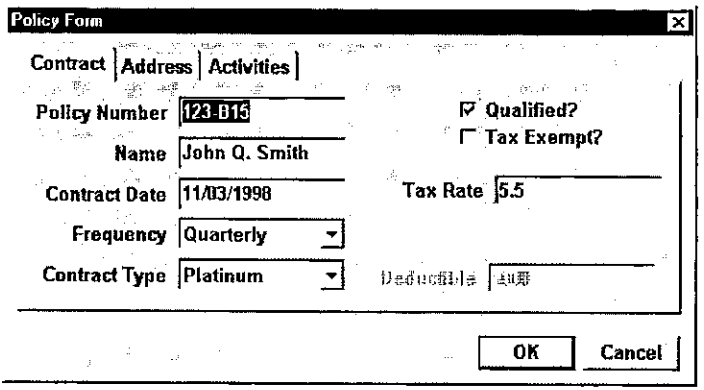
- **Shortcut keys.** Within a ZarkWin form, you can define any number of "shortcut" keys. For each specified keystroke (e.g. Ctrl-T or Ctrl-Alt-R), you can provide an APL expression.

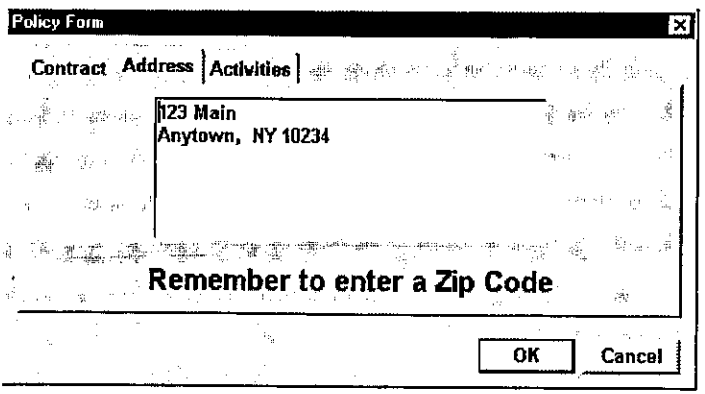
- **Context-sensitive help.** You may provide a "help" document for each Control Object on a form. If the user presses F1 or Ctrl-H, the help document for the Control Object that currently has the focus is displayed in a pop-up window.

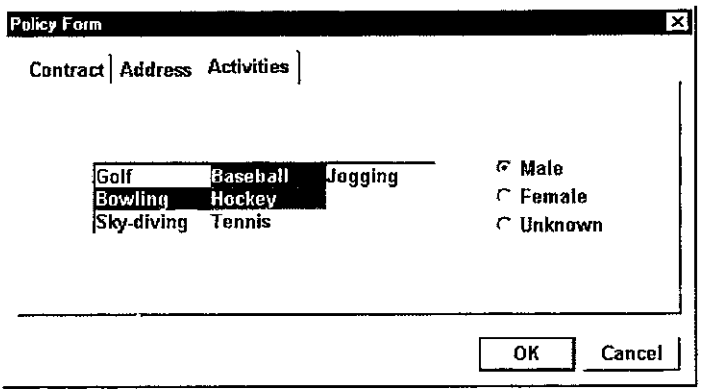
- **Menu bar.** ZarkWin allows you to easily construct a menu bar of arbitrary complexity. The menu items can be checkable or not, enabled or not, visible or not, can have shortcut keys or not, can have associated APL expressions or not, and can be changed dynamically at will.

Illustration

Suppose we want to implement the following input form:







With ZarkWin, the first step is to name each of the input fields. Let's use the following names:

- PNUM Policy number (character vector)
- NAME Policyholder name (character vector)
- CDATE Contract date (yyyymmdd)
- FREQ Payment frequency (1=Monthly, 2=Quarterly, 3=Semiannually, 4=Annually)
- CTYPE Contract type (1=Premium, 2=Gold, 3=Platinum)
- CHKS [1] Qualified? (1=Yes, 0=No)
  [2] Tax exempt? (1=Yes, 0=No)
- RATE Tax rate as a percent (*if* not tax exempt), e.g. 5.5
- DED Deductible amount in dollars (*if* a Gold contract)
- ADDR Policyholder address (newline-delimited character vector)
- ACTIV Activities (vector of any of these: 1=Golf, 2=Bowling, 3=Sky-diving, 4=Baseball, 5=Hockey, 6=Tennis, 7=Jogging)
- SEX Sex (1=Male, 2=Female, 3=Unknown)

The second step is to write a companion function for this form. The purpose of the function is to gather the needed information for the initial presentation of the form, to present the form, to handle any other call-backs from the form, and to save any changes made to the information by the user. If the information displayed on the form is being saved on file, it is the job of this function to read it

from the file when presenting the form, and write it back to the file when the form is OK-closed.

The following is a typical companion function for the above form:

```
     ∇ Z←L POLICY R;ACTIV;CHKS;D;DATA;RATE;SEX;T
[1]    ⍝ Presents the "Policy" form.  Used like this:
[2]    ⍝    formPOL POLICY 'Go'
[3]    ⍝ Other right arguments are callbacks:
[4]    ⍝    POLICY 'Exempt'     if Exempt check box is clicked
[5]    ⍝    POLICY 'Contract'   if Contract Type is clicked
[6]    ⍝    POLICY 'ErrChk'     for error chks when OK-closed
[7]    ⍝
[8]    ⍝ Branch by right argument:
[9]     →('Go' 'Exempt' 'Contract' 'ErrChk'∈⊂R)/L1,L2,L3,L4
[10]    ⎕ERROR 'Unknown argument'
[11]   ⍝
[12]   ⍝
[13]   ⍝ Initial call; read data from file; present form:
[14]   L1:'⍵ DATAFILE' ⎕FSTIE ⍵ ⋄ DATA←⎕FREAD ⍵ 1 ⋄ ⎕FUNTIE ⍵
[15]    T←'PNUM NAME CDATE FREQ CTYPE CHKS RATE DED ADDR ACTIV SEX'
[16]    D←(T ('value' DATA 1)) T zwShow L ⍝ 1=Trigger onClicks
[17]    →(D=0)/0 ⍝ Exit if Cancel-close; else file data:
[18]    '⍵ DATAFILE' ⎕FSTIE ⍵ ⋄ D ⎕FREPLACE ⍵ 1 ⋄ ⎕FUNTIE ⍵ ⋄ →0
[19]   ⍝
[20]   ⍝
[21]   ⍝ Click on 'Exempt'; make tax rate invisible if exempt:
[22]   L2:'RATE' ⍙zw 'visible' (~'2⊃CHKS' ⍙zw 'value')
[23]    →Z←0 ⍝ 0=Don't close form
[24]   ⍝
[25]   ⍝
[26]   ⍝ Click on 'Contract'; enable Deductible only if type 2:
[27]   L3:'DED' ⍙zw 'enabled' (2='CTYPE' ⍙zw 'value')
[28]    →Z←0 ⍝ 0=Don't close form
[29]   ⍝
[30]   ⍝
[31]   ⍝ Error checks at OK-close:
[32]   L4:(CHKS RATE ACTIV SEX)←'CHKS RATE ACTIV SEX' ⍙zw 'value'
[33]    Z←'RATE:10% is the maximum rate' ⋄ →((RATE>10)∧~2⊃CHKS)/0
[34]    Z←'SEX:Only males play hockey' ⋄ →((SEX≠1)∧5∈ACTIV)/0
[35]    Z←'' ⋄ →0 ⍝ Empty if all is well
     ∇
```

The final step is to construct the form as a nested array. This process is begun by typing: zwDef 'formPOL'. Then click, drag, and type until the form looks right.

To run the application, type: formPOL POLICY 'Go'.

Final Comments

This the fourth generation of ZarkWin. While the software continues to evolve, it seems to have reached its final form. As such, it has become everything we hoped for, and more. Our aim was to develop a tool that can be used to implement a user interface in something comparable to the time it takes the user to define and describe it. In other words, we wanted to be able to sit with our fingers poised over the keyboard as the user describes the desired interface. While we knew we could never return to the simple and speedy development of the quad/quote-quad interface, we nevertheless wanted to return to a time when we could wait for the user to make up his mind, rather than the user waiting for us to "program" the interface.

We also wanted to bury our Windows programming knowledge as deeply as possible within sub-functions, so we could purge if from our brains and go back to being the APL programmers we prefer to be.

With ZarkWin, we're pretty close.

After the Lord Mayor's Show

by George MacLeod (george.macleod@simcorp.com)

Introduction

For the benefit of non-British members of the BAA I probably need to explain the title of this paper. Every year the City of London (the financial part of London) elects a new Lord Mayor. His inauguration is celebrated by a procession of golden coaches and floats, many of them pulled by horses. At the back of the procession is the man who clears up after the horses.

I earn my keep as an APL contract programmer, which means that I spend most of my time porting systems that have been around for years or maintaining legacy systems on IBM mainframes. Many of these systems are ten or more years old and have been worked on by a host of people of varying abilities in different styles and different standards, or sometimes, no standards at all.

The advent of APL interpreters that can run in the Windows environment has encouraged the use of APL to build GUI systems. Programming GUI objects using the system functions provided by the interpreter can be a boring job which has to be done with some precision if the result is going to look good. This has caused some programmers to build their own tool kits so as to increase the speed with which GUI programs can be written. These tools are often very effective in speeding up the program writing though they tend to produce rather regimented GUI layouts. The real downside is that these tools become part of the application program. It can mean that layouts are controlled by a number of global variables which if changed can have strange effects all over the system. The other type of tool tends to consist of many small functions which the developer laces together and which are an integral part of the application. The programmer who comes along later has to understand the functions (which generally are not documented) in addition to understanding the application itself.

I have written most of my GUI code using the system functions (`⎕WC`, `⎕WS` etc.) provided in Dyalog/APL. This has been fine but I have always envied the speed with which the developer with a GUI tool kit has been able to create GUI code. If you care about the look of your Windows screen you can spend a lot of time fiddling about with the position and size of objects so that they look good.

GUIBuilder

So I decided to build a meta-language that would do the following:

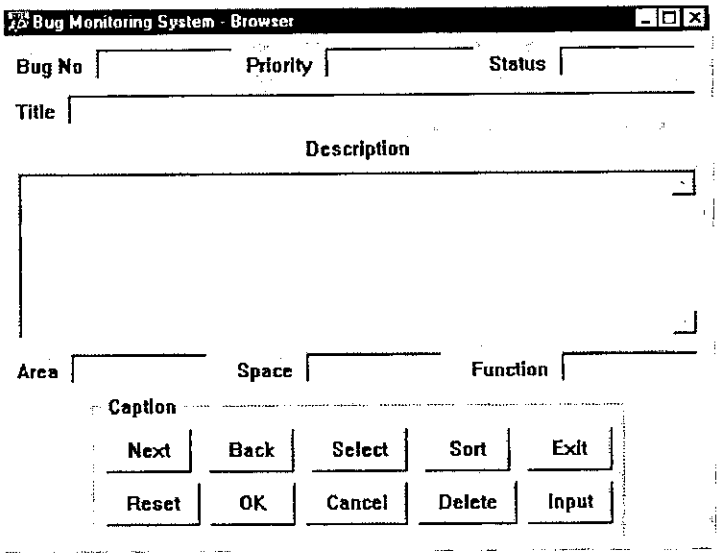
- generate APL code that would produce a populated Window.
- not use any global or local variables in the generated code.
- not use any sub-functions in the generated code.
- compute the Position property of every object.
- compute the Size property of most objects
- generate composite objects to provide a Caption property where one is not usually available.
- allow the height of List, Edit and some other objects to be specified in Rows.
- assist in centring and aligning objects relative to other objects.
- allow the programmer to tweak the layout so that the programmer has final control.

This system I call GUIBuilder. GUIBuilder consists of a number of functions that produce GUI objects, functions involved with locating them on a form and a set of functions to change the spatial relationships between objects (like centring, aligning, spreading etc.). Creating a GUI form by this process generates a function which will produce the form with all its objects and containing only APL system functions to generate them.

Example of GUIBuilder

This is all best illustrated by example.

Let us imagine that we wish to build the following form.




```
[0]  MakeVector
[1]  ⍝ This is an example for Vector
[2]   StartBuilder'Example for Vector'
```

This is the start of a function called MakeVector. The function name is chosen by the user.

The first function invoked is *StartBuilder* which initiates GUIBuilder and whose argument will be used as an initial comment in the created function.

```
[3]   'Bug'Form'C Bug Monitoring System'
```

This creates a Form whose name is Bug and which has a caption of Bug Monitoring System. The C in the parameter indicates that what follows is a Caption (an F would have indicated a Font Name and T a Tip - there are others which are described below).

The size of the Form is not specified as this will be calculated.

When an object is created it becomes the currently Selected Object and the next object will be located relative to it.

```
[4]   'Bug.L1'Edit TopLeft'C Bug No'(0 100)'R 1'
```

This line creates a composite object called *'Bug.L1'* consisting of an Edit object called *'Bug.L1Edit'* and a Label object called *'Bug.L1Label'*. As the Selected object is the parent of the object being created it will be the container of the object being created. Therefore the new object will be inside the form in the top left hand corner. The contents of the parentheses indicate the size of the edit field. The height is specified as 0 because it will be specified later. The width of the Edit object will be 100 pixels.

Normally an Edit object does not have a Rows property but GUIBuilder allows the number of rows to be specified and calculates the height in pixels taking into account the font being used. *'R 1'* indicates that there is one Row.

The Label object will be placed to the left of the Edit object. This is the default.

```
[5]   'Bug.L2'Edit Right'C Priority' 'R 1'(0 110)
[6]   'Bug.L3'Edit Right'C Status' 'R 1'(0 100)
```

Specifying *'Bug.L1'* caused it to become the Selected object and therefore *'Bug.L3'* is specified as being to the *Right* of it. *'Bug.L3'* is to the *Right* of *'Bug.L2'*.

```
[7]   Select'Bug.L1'
```

The next object we wish to place below *'Bug.L1'* so we make *'Bug.L1'* the Selected object.

```
[8]   'Bug.E1'Edit Below'R 1' 'C Title'
```

This is another single row edit field. This time the size is not specified at all. The Row property specifies the height and we will calculate the width later.

```
[9]   'Bug.E2'Edit Below'R 7' 'L Above Centre' ...
      ... 'C Description' 'S Multi'
```

The line defines a multi-line edit field. Again there is no size specification, the height is 7 rows and the width we will calculate later. The L phrase defines the placement of the Label with regard to the Edit object. It specifies that the Label will be centred above the Edit object. The S phrase specifies a Style property and indicates that the Edit object will have a Style property of Multi.

```
[10]  'Bug.E2Edit'Props('VScroll 1)( 'ReadOnly' 1)
```

This line contains the *Props* function which makes it possible to specify additional properties. These properties are added to the □*WC* statement in the Vector function which makes it possible to use properties that may only be specified when the object is created.

```
[11]  'Bug.L4'Edit Below'C Area'(0 100)'R 1'
[12]  'Bug.L5'Edit Right'C Space'(0 100)'R 1'
[13]  'Bug.L6'Edit Right'C Function'(0 100)'R 1'
```

These three objects are similar to 'Bug.L1' through 'Bug.L3'

```
[14]  Select'Bug.L4'
[15]  'Bug.G'Group Below'C Caption'
```

Change the Selected object and below it place a Group object of no specified size

```
[16]  'Bug.G.B1'Button TopLeft'C Next' 'E 30 Exit'
[17]  'Bug.G.B2'Button Right'C Back' 'E 30 Back'
[18]  'Bug.G.B3'Button Right'C Select' 'E 30 Select'
[19]  'Bug.G.B4'Button Right'C Sort' 'E 30 Sort'
[20]  'Bug.G.B5'Button Right'C Exit' 'E 30 1'
```

This places five push buttons across the form. No size is specified as push buttons have a default height and a default minimum width. The width of the Button Caption property is calculated and the width of the button is increased if necessary. The E phrase allows the specification of Events.

```
[21]  Select'Bug.G.B1'
[22]  'Bug.G.B6'Button Below'C Reset' 'E 30 Reset'
[23]  'Bug.G.B7'Button Right'C OK' 'E 30 OK'
[24]  'Bug.G.B8'Button Right'C Cancel' 'E 30 1'
[25]  'Bug.G.B9'Button Right'C Delete' 'E 30 Delete'
[26]  'Bug.G.B10'Button Right'C Input' 'E 30 Input'
```

This defines five more buttons beneath the first five.

```
[27]  'Bug.L1' 'Bug.L3'SpreadObject_H'Bug.E1'
[28]  'Bug.L1' 'Bug.L3'SpreadObject_H'Bug.E2'
```

Now we come to a function that changes things. SpreadObject_H (H for horizontal) changes the width of *'Bug.E1' and Bug.E2* so that they stretch from the left edge of *'Bug.L1'* to the right edge of *'Bug.L3'*. This makes the two wide edit fields line up with the three edit fields above them.

```
[29]  'Bug.L1' 'Bug.L3'SpreadGaps_H'Bug.L4' 'Bug.L5' 'Bug.L6'
```

SpreadGaps_H increases (or decreases) the gaps between objects so that the edit objects at the bottom of the form line up with the other edit objects.

```
[30]  FitContainer'Bug.G'
```

Now we can calculate the size of the group so that it fits nicely around the buttons.

```
[31]  FitContainer'Bug'
```

and now we fit the form to the objects it contains. If you trace the MakeVector function this is the first time that you see what you have created, because up to now the size of the form has been 0 104. (it was specified as 0 0 but the minimum width of the user area of a form is 104 pixels.

```
[32]  CentreHorizontal'Bug.G'
```

This centres the Group containing the buttons.

```
[33]  'Bug.G.B6' 'Bug.G.B10'SpreadGaps_H...
     ...'Bug.G.B1' 'Bug.G.B2' 'Bug.G.B3' 'Bug.G.B4' 'Bug.G.B5'
```

This spreads out the top row of buttons to match the bottom row.

```
[34]  'Bug'PlaceForm TopMiddle
```

This places the Form at the top middle of the screen (taking into account the Screen Metrics so that the Title bar does not disappear off the top of the screen).

```
[35]  'A Extra'Function'Vector'
```

Finally we execute Function which creates the function Vector which we will use in our application to create the GUI. The left argument is optional and specifies a line of code to place at the end of Vector. In this example we will call a function called Extra, or we would if it were not preceded by a lamp. This is what I choose to do as usually the Function Extra (or whatever it is called) has not been written on the first runs of MakeVector.

Finally the function Vector is run so that the final object you will see has been created by the created function.

And this is what Vector looks like.

```
Vector
⍝ Example for Vector
⍝ Created by GUIBuilder from MakeVectorTest
 'DefaultFont'⎕WC'Font' 'System' 16 0 0 0 700 0
 'Bug'⎕WC'Form' 'Bug Monitoring System - Browser'(20 149)(364 502)
    ('Font' 'DefaultFont')('Tip' 'Bug.Tip')
 'Bug.L1Label'⎕WC'Label' 'Bug No'(13 8)(16 48)
 'Bug.L1Edit'⎕WC'Edit'('Posn' 8 64)('Size' 24 100)
 'Bug.L2Label'⎕WC'Label' 'Priority'(13 172)(16 47)
 'Bug.L2Edit'⎕WC'Edit'('Posn' 8 227)('Size' 24 110)
 'Bug.L3Label'⎕WC'Label' 'Status'(13 345)(16 41)
 'Bug.L3Edit'⎕WC'Edit'('Posn' 8 394)('Size' 24 100)
 'Bug.E1Label'⎕WC'Label' 'Title'(45 8)(16 28)
 'Bug.E1Edit'⎕WC'Edit'('Posn' 40 44)('Size' 24 450)
 'Bug.E2Label'⎕WC'Label' 'Description'(72 214)(16 74)
 'Bug.E2Edit'⎕WC'Edit'('Posn' 96 8)('Size' 120 486)('Style' 'Multi')
    ('VScroll' ¯1)('ReadOnly' 1)
 'Bug.L4Label'⎕WC'Label' 'Area'(229 8)(16 29)
 'Bug.L4Edit'⎕WC'Edit'('Posn' 224 45)('Size' 24 100)
 'Bug.L5Label'⎕WC'Label' 'Space'(229 164)(16 40)
 'Bug.L5Edit'⎕WC'Edit'('Posn' 224 212)('Size' 24 100)
 'Bug.L6Label'⎕WC'Label' 'Function'(229 331)(16 55)
 'Bug.L6Edit'⎕WC'Edit'('Posn' 224 394)('Size' 24 100)
 'Bug.G'⎕WC'Group' 'Caption'(256 63)(100 376)
 'Bug.G.B1'⎕WC'Button' 'Next'(24 8)(30 60)('Event' 30 'Exit')
 'Bug.G.B2'⎕WC'Button' 'Back'(24 81)(30 60)('Event' 30 'Back')
 'Bug.G.B3'⎕WC'Button' 'Select'(24 154)(30 68)('Event' 30 'Select')
 'Bug.G.B4'⎕WC'Button' 'Sort'(24 235)(30 60)('Event' 30 'Sort')
 'Bug.G.B5'⎕WC'Button' 'Exit'(24 308)(30 60)('Event' 30 1)
 'Bug.G.B6'⎕WC'Button' 'Reset'(62 8)(30 66)
 'Bug.G.B7'⎕WC'Button' 'OK'(62 82)(30 60)('Event' 30 1)
 'Bug.G.B8'⎕WC'Button' 'Cancel'(62 150)(30 72)('Event' 30 1)('FCol' 128 128 128)
 'Bug.G.B9'⎕WC'Button' 'Delete'(62 230)(30 70)('Event' 30 'Delete')
 'Bug.G.B10'⎕WC'Button' 'Input'(62 308)(30 60)('Event' 30 'Input')
⍝ Extra
```

This GUI code has been produced without having to specify any position properties and very few size properties. There is no detritus and the code is very easy to understand.

If the original developer has to make changes then this can best be done by modifying the meta-code and re-running it.

There are a number of functions not described in this paper but they mostly do similar things to those described.

One exception is *Ratchet*, which moves an object in a specified direction.

```
'Bug.G'Ratchet Down 10
```

This code would move the Group object down by 10 pixels. It is important that such a line of code is placed in the correct place in the function. Clearly it has to go after the line creating the group and putting it immediately after is a good idea. It must be before the line that fits the Form to its contents or the Form will be too small. The function Ratchet is provided so that the programmer has the final say in the location of any object.

Final Thoughts

I started using GUIBuilder before it was finished and thoroughly tested. Every time I made a window GUIBuilder fell over or showed that it needed additional functionality. Finally I built a window that went smoothly and writing the meta-code was extremely rapid. I am very pleased that I built the system and will use it in all future work.

GUIBuilder is not yet complete as not all objects are handled. I would like to handle PropertySheets and PropertyPages. These are interesting as the size of all PropertyPages is controlled by their parent PropertySheet. Unfortunately specifying the Size property is unreliable at this time.

Finally, best of all, I will not be providing material for the man who follows the horses!

If anybody is interested in trying GUIBuilder then contact me on:

george.macleod@simcorp.com

Taking the *Migraine* out of *Migration*

by Walter G. Fil

This paper was presented at the APL2000 Users' Conference, Orlando, Nov. 1999

Interesting Times

"May you live in Interesting Times" - goes the old Chinese curse. Well, Interesting Times are finally here. With Y2K issues hopefully safely over, but between the Redmond-mandated disappearance of DOS and 16-bit Windows, to the disappearance of numerous European currencies to give way to the Euro, life could not be more interesting. Our customers have quite a bit to do. With the introduction of Windows NT a few years ago, our customers somehow can't help but migrate - not an unreasonable decision in the face of remaining current. Nevertheless, one of the things which simply does not work reliably under Windows NT is APL+DOS, previously known as APL*Plus II/386.

And for good reason. The world has changed considerably from those heady days in the late 1980s when the Intel 386 processor and APL*Plus/II ware first unleashed. From the point of view of an APL programmer, finally, here was a platform which could realistically handle mainframe-sized applications which would run at reasonable, or compared to VSAPL on an overloaded VM machine, excellent speed. Nonetheless, with the astronomical growth of Windows usage, APL applications could only survive as Windows applications, thus APL*Plus/III, forerunner to APL+Win, was born. Among other things, the mechanics of managing large amounts of memory, amounts well over the long-forgotten 640K barrier, were different and not entirely compatible between Windows and DOS applications. Windows took over control of all peripherals, everything from the mouse to the printer. Where previously these sorts of things were handled in applications, Windows now usurped this responsibility. The fact that some DOS applications worked in Windows at all was a minor miracle.

DOS, Windows 3.1, and everything in between were 16-bit operating systems. Windows NT is largely a 32-bit operating system. Windows 95 and Windows 98 are in fact bridges between the 16- and 32-bit worlds, containing a compromised mix of features, designed to facilitate the incremental migration of your applications from one world to the other. As for DOS emulation, Windows 98 contains enough vestiges of DOS and the 16-bit world for APL+DOS to run reliably. Windows NT does not.

Welcome to Windows

Windows requires numerous changes to your applications. First, there are the Windows themselves - volumes on GUI design and programming are the things that line bookstore shelves. The Windows GUI offers an incredible choice of controls and various gadgets and incredible flexibility for the finished product. Not surprisingly, compared to the old way of doing things, it requires considerable programming effort. With ODBC, a completely new perspective on how and where to put your application's data is available. Coupled with the SQL database of choice, your data is no longer subject to entombment in APL component files - in an SQL database, it is available for other applications to use. No longer is application data stuck in something some people might call "proprietary". Finally, the mechanics of true Windows printing are rather different, in fact, an application developer might opt to have another product handle complex printing tasks. With Windows, *connectivity* between different products allows your APL applications to be true team players with respect to everything else running on the computer.

When you look at different APL systems and the way they evolve with time, you seldom see language features taken away. As Windows is conceptually superior to DOS, it was not surprising that many DOS-specific features of APL+DOS had been removed. Some of the things, like the `⎕G` suite of graphics functions, were easily and readily rewritten with Windows. Other things, such as the `⎕WIN` suite of functions, are somewhat more difficult to replace, the ideal solution being to rewrite the application in Windows. We considered the lack of a `⎕WIN` replacement to be the only real obstacle in migrating APL+DOS applications to APL+Win when it needed to be done expediently. Other changes, such as the removal of `⎕ARBIN` as a facility to do printing, were minor, and often required a small amount of rethinking. One application used Hewlett-Packard-specific escape sequences to change printer fonts - this was easily changed with simple Windows programming.

DOS-style APL applications brought with them a style of full screen input which was implemented with the `⎕WIN` suite of functions: these included the functions `⎕WGET` and `⎕WPUT` for reading and writing text and attributes to the screen, `⎕INKEY` for accepting characters, and `⎕WIN` and `⎕ED` for full featured data display and entry. This DOS style of full screen input and the suite of system functions themselves have been rendered obsolete by Windows. Consequently, they have not been included in any version of APL+Win. Clearly, the Right Thing to do is to rewrite any application making heavy use of obsolete language features.

There are many times when the Right Thing just is not possible to do. Either rewriting the application in question is not economically feasible or justifiable, as the plan is for it to disappear soon. Or maybe it's just too hard to do in a short time. Or maybe there's not the time to do it now, with the Euro just around the corner. But whatever the reason, the application must run under Windows NT, and soon! For whatever excuse, our solution was to implement a usable subset of the `⎕WIN` suite of functions for APL+Win, collectively known as Qwin.

Our philosophy is, where applicable, to try to accomplish the migration incrementally. Do the work a piece at a time, such that if anything goes wrong, you are only one step away from something that worked. Our feeling is that all systems migrated with the aid of Qwin should eventually be rewritten to use a proper GUI interface. Good GUI design and programming is a lot of work and could easily be the single most difficult item on the migration agenda. Qwin was designed to help you get your application working now, giving you the freedom to concentrate on the GUI later.

Finally, a good migration will guarantee the success and staying power of your application. Your application contains a lot of knowledge, collected over time. When you migrate your application from APL+DOS to APL+Win, you have at your disposal a large number of new facilities that can add incredible value to your application. On the surface, there is the GUI of Windows, but don't forget about ODBC, Open Database Connectivity, or TCP/IP, providing access to services on your network or over the Internet continents away. Products like Microsoft Excel and Word can assist in document preparation and data import and export. Your APL application can re-emerge in Windows NT and look and feel exactly like a Windows application should. The heart of the application, the valuable application knowledge, the code which comes up with the *right answer*, could remain relatively undisturbed.

Qwin replaces some missing functions

Looking at the APL interpreter when migrating from APL+DOS to APL+Win, the core features of the language have hardly changed. With the exception of support for long file names, the `⎕F` file functions are fundamentally unchanged. Files *written* with APL+Win can be *read* with APL+DOS, providing backwards compatibility that offers some interesting options for running your application in parallel during its migration. Native file support is the same. Event handling is unchanged. `)EVLEVEL`, the evolution level, is set higher by default, but that too can be set lower if needed.

However, several groups of functions are conspicuously missing. First and foremost are the suite of functions to do the DOS style of full screen input, such as ⎕*WIN*, ⎕*WGET*, ⎕*WPUT*, and so on. Included in this group are the functions to do keyboard input, ⎕*INKEY* and ⎕*INBUF*. The functions to allow graphics are gone but do not pose a significant barrier to migration since their functionality is relatively easily duplicated with the native Windows GUI.

Some functions are there but need extra attention - ⎕*CALL* has been carried over from APL+DOS to APL+Win, but its argument - the representation of a compiled program - will have to be changed as the original program will have to be recompiled for the new environment. Many utility functions which use ⎕*CALL* have 100% APL replacements. ⎕*CALLs* are accompanied by ⎕*STPTR*, a clear indication of a system dependency.

For our migration work, the lack of the ⎕*WIN* suite of DOS full screen function was the largest single obstacle to getting APL+DOS applications over to Windows NT, where they would be safe for at least the next few years. Our workload was far too much to even consider rewriting the applications in Windows, and the customer's priorities were more oriented toward instant ODBC connectivity. The solution was clear - although the ideal thing would be to rewrite the application with Windows, the lack of time and funding dictated the regressive yet practical approach of emulating ⎕*WIN* in APL+Win.

False Starts

When we first started development of Qwin, we were of the opinion that we had to use a lower-level language to do the job, on the account of execution speed. We knew enough Windows programming to design a form, execute a callback, and so on, but little else. The first goal was to implement the ⎕*WGET*, ⎕*WPUT* and ⎕*INKEY* functions, that is, be able to do rudimentary screen painting and keyboard input.

The first attempt was with the Microsoft Visual Basic language. We found the environment friendly enough, but in the end, cumbersome, as VB was designed to insulate the developer from details of Windows programming to speed application development.

Our second attempt was with Microsoft Visual C++, where the goal was to produce a custom control (.OCX) which was to be integrated with an APL+Win *Form* object. In the end, we had a working .OCX control, however a few things became very clear. Firstly, development in this environment was painfully expensive. In the end, we had a facility to paint a screen and accept keyboard

input, but to add any complex functionality, such as *⎕WIN*, *⎕ED*, or *⎕EDIT* was going to be much too expensive and would take too long with this approach. The productivity we had come to expect with using APL all those years just wasn't there, even with an extremely experienced and competent developer doing all of the C++ work. Secondly, and perhaps more importantly, the isolation of the .OCX control from the APL environment was technically inappropriate. For *QINKEY*, the emulation of *⎕INKEY*, the control returned key scan codes but required a high level of interaction , i.e. a large number of repetitive calls, to accomplish this. To be able to support a facility like *⎕MOUSE* would require even more close interaction. A KeyPress or a MouseClick event needed to be handled somewhere locally in the APL environment, not in a distant .OCX control, only to be passed back to APL to be dealt with. In short, all of this really needed to be happening much closer to the action in the APL workspace.

In the end, we chose to stick with a 100% APL solution. One problem which popped up at a customer's site was that installing and registering the .OCX control required more privilege than the user had. A needless complication was the requirement of configuring and using installation software such as InstallShield for what amounted to a small component. It was clear that fewer disparate moving parts would be better.

During a talk in the summer of 1999, Eric Baelen, president of APL2000, demonstrated the APL+Man game, an example of where APL+Win is convincingly used in a *real time context* with excellent all-around performance. Qwin is very much a *real time* programming exercise. Best of all, with APL, Qwin is easy to extend and maintain, as we do not rely on third party support of non-APL software.

The Window

Just how hard can it be to write a function that displays a 24 by 80 window of characters? First of all, it's a 25 by 80 window or a 50 by 80 window, depending on the video mode used in DOS. Secondly, it's only part of the problem - keyboard input has to be done as well. Third, absence of little details such as a blinking cursor, render the emulation unusable. A little knowledge is a little dangerous, but fortunately for us, we had no real knowledge of grass roots Windows programming whatsoever.

The Qwin main window is a program not entirely unlike the APL+Win function editor. With syntax colouring, different items in your APL programs are displayed in different colours depending on the type of token the text represents. Turning insert mode on or off changes the style of the blinking cursor. Not to be

taken for granted, these features are implemented by programs working behind the scenes of the APL+Win function editor.

As our Windows programming experience was limited to dialog boxes and the like, our first idea was to utilise the existing Windows controls, such as Edit and RichEdit, to handle the application display area. The Qwin window would contain a single edit field which occupied the entire window. This couldn't work for several reasons. First, neither the Edit nor the RichEdit controls allowed you to arbitrarily change both the foreground and background colours of the text displayed. With a RichEdit control, you could change the foreground, but not the background, colours of the text. With an ordinary Edit control, you could not vary the colours at all. The blinking cursor was inflexible to the extent that only one shape was available, a vertical bar, and it could not be made to disappear. Any edit control had simple program logic behind it, which was to accept a keystroke and insert it wherever the cursor was, which could be at the end of the last character in the field. Although you could alter this somewhat with ⎕*WRES*, for use with Qwin, character entry and display are best completely divorced from each other.

In the end, the Qwin main window is a featureless window with no controls at all. Text is written, or more accurately, painted, a line at a time, with complete, explicit control of the foreground and background colours. The Windows font used with Qwin is the same as the Windows font supplied with APL+DOS, with a few minor modifications. This is a bitmapped font where all characters are exactly the same width - this means that characters can be drawn a "strip" at a time. This also means that the APL code to repaint a screen is fairly simple and efficient. The APL+Win system also supplied a bitmapped font which is typically used for the session manager and the editor – it is usable except that it is missing many of the line drawing characters!

Ordinary character input is accomplished collecting keystrokes in a buffer, then using them when needed. The Qwin main window invokes a callback for every keystroke. These are in turn mapped into the familiar numeric codes returned by ⎕*INKEY* and placed into a buffer, or discarded if invalid. When the next character is needed, the first character is removed from the buffer. When a character is needed but the buffer is empty, the ⎕*INKEY* emulation loops, with the help of a generous ⎕*WGIVE*, until a character appears in the buffer. Qwin translate tables were reconstructed from the documentation available from APL+DOS. Although a few differences exist between the APL+DOS and Qwin keyboards owing to various differences in low level keyboard handling in DOS, Windows NT, and APL+Win, the Qwin keyboard is a faithful reproduction of the APL+DOS keyboard.

Other details, such as cursor handling, are accomplished with ⎕*WCALLs* to Win32 API functions. The cursor, known as a *caret* in Windows parlance, is called into existence when the Qwin window is active, moved as characters are typed, and is wiped out when the Qwin window is inactive. When Qwin is first invoked, the Caps Lock and Num Lock keyboard states are checked by calling the appropriate API functions. These things are not supported in the various ⎕*WI* properties, and since this type of programming is probably so far away from the mainstream of Windows programming in APL+Win, language developers have not included these sorts of features.

On the surface, the Qwin main window would appear to be a modal dialog box, meaning one which causes all application activity to cease until the input in the dialog box is complete, at which time it goes away. The problem is that the Qwin main window cannot go away until the entire application is finished. The dialog box you see when you finish editing a function reminding you that the text of the function has changed and asking if you want to save the changes is a canonical example of a modal dialog box.

With details of screen painting and keyboard entry out of the way, the remainder of the functions are the most difficult ones, namely ⎕*WIN* and ⎕*ED*. ⎕*WIN* is the backbone of many complex screen entry programs, it provides the ability to break up a screen into separate fields, validate input, change the colour of a field depending on whether it is active, and so on. ⎕*ED* allows you to browse a character matrix in your application, or even have a custom editor session going, possibly integrated with other fields entered by ⎕*WIN*. Further, utility functions such as the screen design workspaces supplied by Zark Incorporated and delivered with the APL+DOS system combined the functionality of ⎕*WIN* and others functions to provide APL application programmers a truly comprehensive set of building blocks which are difficult to replicate with native Windows features. ⎕*WIN* and ⎕*ED* display their fields and take care of, keystroke by keystroke, entry into fields, moves between fields, and the return codes upon exit of the respective function.

Migraine-free Migration Strategy

Our APL application migration strategy, used with many different combinations of APL systems, is:

- Where it makes sense, start making changes in the old environment, as soon as possible.
- Do as little as possible to make the application work exactly as before in the new environment.

- Add new features, one by one, to the application in the new environment, taking care to make sure one task is finished before starting another.

When migrating from APL+DOS to APL+Win, many of the old problems related to moving the APL code, variables, and data from the old environment to the new environment simply disappear. Workspaces from APL+DOS can be freely)LOADed, plus the component file system is identical, to the extent that file system data written with the newer APL+Win can be read with the older APL+DOS. This feature makes it much easier to run your application in parallel.

In a more traditional migration, such as from a mainframe APL2 system to APL+Win, you still have the problem of physically moving the code and data from one place to the other. Once you have the code over, there are countless language differences which need to be detected, isolated, and solved. Some facilities exist for doing some of these jobs and are usually easy to use, but nothing beats the ease of movement from APL+DOS to APL+Win.

Back to the strategy - when you know that certain conditions will exist in the destination APL system, you can start making changes as early as possible to accommodate them. If you know that eventually your system will read its data from an SQL database, and that APL variables in the workspaces will correspond to SQL field names, but that a few of these variables are SQL reserved words (such as SELECT, INT, UPDATE, FLOAT, and so on), change them now. If you are moving an APL2 (or VSAPL) workspace which uses the system function ⎕*EA*, and nearly all uses are of the trivial case, typically used to supply a default value to the left argument of a dyadic function when used monadically, change it now. With APL+DOS, maybe now is the time to change your code so that it operates with the highest, rather than the lowest, evolution level in APL+Win. At this stage, you are probably more familiar to the old environment than the new and making changes would be less risky. The point of this is to increase productivity and reduce overall migration stress by carefully spreading the activities over time.

When you run into a non-portable feature, it usually is best to code a cover function to provide duplicate functionality in the destination system, at least initially. Cover functions have the following design advantages:

- You can make mechanical replacements in the destination system
- You can use it over and over again
- All of the knowledge is contained in one function
- It is easy to prove it correct

- It is easy to fix
- Changes to the cover function are not intrusive with respect to the rest of the application

The only design disadvantage to using cover functions in this manner is that they may be slightly slower than a more direct solution. Our Qwin product is built on the design principle that it should be possible to `)COPY` the Qwin functions into your workspace, make the appropriate mechanical replacements, make a few changes, and be pretty close to finished.

Once the application, in its preliminary form, has been moved to the new environment, the only thing to think about and do is *to get it to work*. Now is not the time to add GUI features or ODBC support. Instead, the application should be tested thoroughly to uncover whatever surprises may be lurking. If circumstances permit, the application should even be put into production. This is also not to say that you should work slowly, just that you should focus only on one problem at a time before moving on to the next one.

One of the pleasant surprises which awaits you in the new environment is a significant increase of the available workspace – `⎕WA`. Back in the early days of the 386, it was not uncommon to have just 4 or 8 *megabytes* of memory in your computer, leaving you with a corresponding number of megabytes of workspace. That seemed like a lot, especially when you just moved your application from APL+PC or even some of the timesharing services. Later, when Windows arrived on the scene, 16 megabytes was not an unreasonable amount of memory to have. Today, for Windows NT, 64 megabytes is the lower boundary for an acceptable amount of memory for a typical NT computer. As for your application, the added memory can make your programming job easier and more productive. Your application will run faster. Be sure to use it.

Our migration toolkit consists of a search and replace facility which goes through all of the functions in a workspace, and a collection of `Q...` functions we've written over the years to emulate many different functions from many different systems. Good migration productivity has amounted to:

- Understanding as much as possible as early as possible about the differences between the two systems
- Having the tools to isolate calls usage of certain primitives or code fragments and to replace them with calls to functions
- Having those replacement functions on hand

Since Qwin functions directly replace □-functions, converting applications is fairly straightforward. We supply a migration function with the Qwin functions. There are, however, a few differences which may have to be dealt with on a case-by-case basis. As the majority of our applications we had experience with used the Zark-supplied screen functions, our primary emphasis was to insure that the utilities worked correctly. Also, the SITDEMO workspace supplied by Zark offers a convenient migration demonstration.

To modify your application to use Qwin, you have to:

- Run the supplied conversion program to make all Qwin-related replacements
- Insert code in the application to start and stop Qwin
- Test the application
- Check for and deal with known problems and special cases, such as setting *□MOUSE* to iota 0, hard-coded *□AV* references to line drawing characters, and so on.
- Test some more

Qwin supplies replacements for the following system functions:

□ARBIN (partial)	*□CRT*	*□DL*	*□ED*	*□EDIT*
□FI	*□INBUF*	*□INKEY*	*□PEEK*	*□POKE*
□SOUND	*□VI*	*□WGET*	*□WIN*	*□WKEY*
□WPUT				

Qwin respects the settings of the following variables:

□CURSOR	*□MOUSE*	*□WINDOW*	*□WKEYS*

Using Qwin

As Qwin runs in a separate execution window, it must be initialised at the time the application is started and cleared at the time the application is exited. This is accomplished with the *QWININIT* and *QWINCLOSE* functions. These would be typically placed in the function started by *□LX*. For developing and maintaining these Qwin applications, life is made much more agreeable by the fact that the

APL session window is separate from the Qwin execution window. Any errors are visible and not artistically fused into the application display.

Qwin does not add any additional global variables to application execution other than variables which were system variables in APL+DOS. All Qwin variables are neatly tucked away inside Qwin as user-defined properties and will not interfere with variables inside of your workspaces or applications.

An important difference of Qwin versus APL+DOS is that the display screen is refreshed for display only at the time it is actually needed, with is usually immediately before a *QINKEY*. In application programs, screens may be built up by successive calls to *QWPUT*. As a performance measure, no actual screen output takes place during a *QWPUT*. This technique fails, for example, when a message is to be displayed with *QWPUT* for a short duration then removed. The solution is to use *QDL*, which first forces a refresh of the screen, then behaves like an ordinary ⎕*DL*. Users of IBM's GDDM and numerous AP124 versions should remember similar behaviour where by default, the screen would be refreshed immediately prior to character input. Only by executing an FSFRCE command in GDDM, or using an immediate write in AP124, would you get the desired effect of seeing everything on the screen prior to any input.

A second important difference is that system variables such as ⎕*CURSOR* and ⎕*WINDOW* in APL+DOS are ordinary variables named *QCURSOR* and *QWINDOW* in Qwin. True system variables are subject to *passthrough localisation* whereby values in a function which localises these names are inherited from the function which called it. Ordinary variables are not subject to passthrough localisation and have no value unless assigned. Qwin attempts to duplicate passthrough localisation for its key variables by assigning the last values it has seen if it encounters what would have otherwise been a VALUE ERROR. If a function reassigns one of the key Qwin variables, then immediately calls a function with the key Qwin variables localised, those latest values will not be propagated to the called function. Again, the solution is to use QDL 0, which has the beneficial side effect of refreshing cached variables for emulating passthrough localisation.

Replacements for ⎕*PEEK* and ⎕*POKE* are supplied with Qwin. Strictly speaking, you will be able to *QPEEK* and *QPOKE* into a 1000 element integer vector but relatively few of these locations have any meaning or effect. Reading and writing into the workspace, i.e. *QPEEK* and *QPOKE* with *QSEG* set to something other than iota zero has no effect.

Incidentally, the ⎕*INKEY* function exists in APL+Win. This ⎕*INKEY*, however, is separate from the *QINKEY* supplied in the Qwin package.

Using Windows 95 and Windows 98

Qwin was developed with only one goal in mind – to facilitate emergency migration of legacy APL+DOS application to APL+Win under Windows NT. Windows 95 and Windows 98 both run APL+DOS reliably, thus we consider them a stepping stone to a proper migration to Windows NT. There is considerable appeal and practical migration value, however, in having APL+DOS and APL+Win applications running side-by-side on the same computer. Our priority is to concentrate on Windows NT first, and Windows 95 and Windows 98 second.

The reality of the situation is that although all of the 32-bit Windows platforms share a common application programming interface, they have subtle differences. The application programming interface is the library of Windows programs available to APL programmers using the *⎕WCALL* system function. Workarounds exist for many differences, but incorporating them may compromise the quality of the Qwin product. Known differences include:

- Cursor keys repeat in 95 and 98 but not in NT
- Differences in KeyUp and KeyDown
- Different Shift+Ctrl+Alt combinations with Caps Lock will set the Shift Lock state differently
- In NT, *⎕CMD* operates synchronously, i.e. *⎕CMD* waits until the other program is done; under 95 and 98, *⎕CMD* returns immediately
- *QSOUND* does not work

Our tactic with Windows 95 and 98 is to concentrate on Windows NT, while attempting to maintain as much functionality as is practical to include in Windows 95 and Windows 98.

On to Printing

One other major topic of migrating to APL+Win is the issue of how to do printing. In APL+DOS, the technique was usually something like this:

- Prepare text to be printed as a character matrix
- Add titles, headings, page numbering, etc.
- Add a carriage return and line feed to every line

- Write this text to a DOS file and issue PRINT command, or write directly to LPT, or use 3 ⎕*ARBIN*

Qwin includes a *QARBIN* function which emulates some of the features of ⎕*ARBIN*. A replacement for 3 ⎕*ARBIN* is provided, and can send text containing device-specific codes (such as PostScript or HPCL) to your default Windows printer without interference from whatever printer driver you may have installed.

Due to the wide variety of styles of printing, this problem does not easily lend itself to mechanical solutions. To use Windows facilities, APL code which does printing will have to be examined and replaced manually. If you can modify your application while you're still in the old system to use a small number of *print utilities* for printing a page of output at a time, then this becomes much less of a problem in the new environment. Windows printing is page oriented. If your application already prints pages of text through a cover function in APL+DOS, then your task is to write a replacement printing function in APL+Win. If it doesn't but can, the time to do this is now.

Strictly speaking, you can still write your text to be printed to a file then print it with the PRINT command, or write the text directly to LPT with ⎕*NAPPEND*. However, you lose the ability to route your printout to the Windows printer as it is set in the Windows Control Panel. Also, on some networks, it will take some work to have the network software "capture" whatever is being sent to LPT1 and to redirect it to wherever it is supposed to go. Lastly, a ⎕*ARBIN* function is supplied with APL+Win, but has few of the features of the ⎕*ARBIN* function from APL+DOS.

If you have printing which requires use of device-dependent codes to change fonts, switch to bold, change sizes, and so on, you should consider farming out your printing to Microsoft Excel and Microsoft Word. With the advent of APL+Win Version 3.5 and ActiveX, it is possible to have either of these products silently do your printing for you. Microsoft Excel is extremely well suited to doing tabular-style reports, and best of all, you have the option of delivering machine-readable output to your users instead of or in addition to paper hardcopy. Microsoft Word is suited to letter writing, where you have letters which consists of blocks of text where your application "fills in the blanks".

Don't Bother

Certain features of APL+DOS and other APL systems do not lend themselves well to emulation. In APL+DOS, you can write directly into the workspace - ⎕*PEEK* and ⎕*POKE* with ⎕*SEG* set to something other than iota 0 immediately come to

mind, where you directly peek and poke into you computer's memory. Another DOS casualty is the ⎕*CALL* function where an integer representation of a compiled routine is supplied as its argument. True, ⎕*CALL* does exist in the new environment, however the compiled routine must be recompiled and rebuilt to work. Unless execution speed is critical, these might be best to replace with a straight APL solution. ⎕*NA* exists in a variety of systems, but they are all different.

Conclusion

When migrating APL+DOS applications to APL+Win, things should go pretty smoothly as the core language is very compatible. Peripheral features of APL+DOS will be different or not present in APL+Win, features such as DOS-style windowing, printing, and interfaces to low-level facilities. In our migration work, we have isolated the ⎕*WIN* suite of functions as a realistic chunk of APL+DOS which could be duplicated in APL+Win. Qwin enabled us to move quite a few applications to Windows NT which could have otherwise been discontinued, abandoned, replaced, rewritten, or would have caused problems for customers who otherwise had no plans to maintain DOS, Windows 3.1 or Windows 95/98 workstations on their Windows NT networks. Other missing language features or different migration items are smaller in scope and often don't lend themselves to mechanical replacement – these you have to develop. Finally, once your application is working in Windows, a wide range of services is available to you, the application developer, which will enhance your productivity and improve the perceived quality of the application.

And remember, Qwin, as described here, is experimental in this version of APL+Win; it may be changed, replaced, or removed in future versions.

References

[1] Richard J. Busman, Walter G. Fil, and Andrei V. Kondrashev, *Recycling APL Code into Client/Server Applications*, APL95 Conference Proceedings, *APL Quote-Quad*, Volume 25, No. 3, 1995

[2] Gary A. Bergquist, *The Zark Library of Utility Functions*, APL99 Conference Proceedings, *APL Quote-Quad*, Volume 29, No. 2, 1999

[3] *APL*PLUS II/386 Version 4 Reference Manual*, STSC Inc., 1992

[4] *APL+Win Version 3 Reference Manual*, APL2000 Rapid Application Development, 1997

[5] *APL+Win 3.5 Update Manual*, APL2000 Rapid Application Development, 1999

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

Vector Jan 2000: Utility Corner (pp. 27-35)

From: H.J.Veenendaal (H.Veenendaal@wxs.nl) 16th March 2000

Remarks by the solution on disk: (workspace VECTOR)

1. I did not understand the purpose of the mixed use of *ANALYZE* and *ANALYSE*.
2. I used *ANALYSE* for testing of *ANALYSE1* (= modified given function with same line-numbers) (this makes testing much easier and you don't create extra global variables in the workspace)
3. The given function doesn't work if *FNAME* has no local variables!!! (but see for my correction the lines [42] and [55]). The correction in line [31] accounts for a function that doesn't exist or is locked.
4. In line [60] a character 's' must be changed in 'z' (but I used ⎕*AV*)
5. A display of FNS is not given because it concerns a analyse of *FNAME* and not of the workspace!
6. I removed duplicate names from IDS (a SORT should also be useful) by adding lines [86] and [87].
7. The function *SUBFNS* gives the indices for the subfunctions in FNS and uses the subsubfunction *BIJMAT*.
8. You should (for completeness) have added a similar analyse for global variables used in *FNA*
9. I use APL*PLUS SE.
10. The workspace VECTOR contains also the function *DOEALL* (using TEXTREPL for 'MMM').

You can test *ANALYSE* for alle functions in the WS by excecuting :

```
(⎕NL 3) DOEALL 'ANALYSE  ''MMM'''
```

[Second letter dated 18th March]
Herewith an improved version of my workspace 'VECTOR' (my letter dd. 16/3)

The improvements concern:

A display of FNS is added to function *ANALYSE*

An error-message is given by function '*ANALYSE*' for function-names shadowed by local variable-names, locked functions and not existent fns.

(see *ANALYS1* [0] and [31] for corrections concerning your *ANALYSE*-function)

A sort of the IDS-list is added to *ANALYSE1* (see [88]).

Function *BIJMAT* (called in *SUBFNS*) is renamed in *INMAT*.

A part of my function *DOEALL* was (very) superflous and is now removed.

Remarks:

After *ANALYSE 'TEXTREPL'* the identifier *TEXTREPLvOBJ* is missing owing to '⍬'!.

Line [87] of *ANALYS1* doesn't work (in APLS*PLUS SE) for large functions with many identifiers, do you have a better solution?

NB: You can test the *ANALYSE* functions for all functions in the workspace by executing:

```
(⎕NL 3) DOEALL 'ANALYSE  ''MMM'''
```

H.J.Veenendaal
Barnsteenhorst 304 2592 ER
Den Haag (NL)

Code Listings (see www.vector.org.uk/v171/hjv.zip)

```
      ∇ ANALYSE FNAME;FNS;RESULT;ARGS;LOCALS;LABELS;IDS;IDS1;COLS;CR;
          HDR;I;LEN;LET;MAX;N;NUM;OK;START;VC;WID
[1]   ⍝ ANALYSE the identifiers within the function named in FNAME.
[2]    ' '
[3]    FNAME,':'
[4]    (1+⍴FNAME)⍴'='
[5]    →(0=1+⍴⎕CR FNAME)/ERR
[6]    ANALYSE1 FNAME
[7]    'FNS:    The names of functions in the workspace'
[8]    '---'
[9]    FNS
[10]   ' '
[11]   'RESULT: The name of the result variable (0 rows if none)'
[12]   '------'
[13]   RESULT
[14]   ⍞(0=1+⍴RESULT)/'''[none]'''
[15]   ' '
[16]   'ARGS:   The names of the argument variables (0, 1 or 2 rows)'
[17]   '----'
[18]   ARGS
[19]   ⍞(0=1+⍴ARGS)/'''[none]'''
[20]   ' '
[21]   'LOCALS: The names of the variables localised in the header'
[22]   '------'
[23]   LOCALS
[24]   ⍞(0=1+⍴LOCALS)/'''[none]'''
[25]   ' '
```

```
[26]    'LABELS: The names of the labels'
[27]    '------'
[28]    LABELS
[29]    ⍕(0=1↑⍴LABELS)/'''[none]'''
[30]    ' '
[31]    'IDS:    The names of all identifiers within the body of
          the functions'
[32]    '---'
[33]    IDS
[34]    ⍕(0=1↑⍴IDS)/'''[none]'''
[35]    ' '
[36]    'SUBFNS: Indices of subfunctions'
[37]    '------'
[38]    SUBFNS
[39]    ⍕(0=⍴SUBFNS)/'⎕CURSOR[1]←¯1+⎕CURSOR[1]'
[40]    ⍕(0=⍴SUBFNS)/'''[none]'''
[41]    ' '
[42]    'SUBFUNCTIONS:'
[43]    '-------------'
[44]    FNS[SUBFNS;]
[45]    ⍕(0=⍴SUBFNS)/'''[none]'''
[46]    →0
[47]  ERR:
[48]    ' '
[49]    'ANALYSE NOT POSSIBLE FOR: ',FNAME
[50]  ⍝ ANALYSE: H.J.Veenendaal 18/03/00 11:29
      ∇

      ∇ ANALYSE1 FNAME;⎕IO
[1]   ⍝ ANALYSE the identifiers within the function named in FNAME.
[2]   ⍝ The ANALYSE1 function creates six global character variables
[3]   ⍝ with one left-justified name per row:
[4]   ⍝ FNS, RESULT, ARGS, LOCALS, LABELS and IDS (see also ANALYSE)
[5]   ⍝
...
[27]  ⍝
[28]   ⎕IO←0
[29]   FNS←⎕NL 3
[30]   COLS←1↓⍴FNS
[31]   →((0=1↑⍴CR←⎕CR FNAME)∨(3≠⎕NC FNAME))/0 ⍝ Quit if FNAME not available
[32]   WID←⍴HDR←CR[0;]
[33]  ⍝
[34]   CR←1 0↓CR
[35]   LEN←('←'∈HDR)×HDR⍳'←' ⍝ Length of the result-var if any
[36]  ⍝ Result is 0 or 1 row matrix (0 if to long)
[37]   RESULT←(((LEN≤COLS)∧×LEN),COLS)⍴COLS↑LEN↓HDR
[38]   HDR←(LEN+×LEN)↓HDR
[39]   HDR←(+/∨\⌽HDR≠' ')↑HDR ⍝ Delete trailing blanks
[40]   LEN←HDR⍳';'
[41]   ARGS←LEN↑HDR
[42]   HDR←(LEN↓HDR),';' ⍝ Drop syntax leaving semicolon!!!!!
[43]   I←(ARGS=' ')/⍳LEN
[44]   N←⍴LEN←(I,LEN)-0,I+1 ⍝ Length of names in syntax
[45]   MAX←⌈/LEN ⍝ Length of longest name
[46]  ⍝ Convert to matrix:
```

```
[47]   ARGS←(N,COLS)↑(N,MAX)⍴(,LEN∘.>⍳MAX)\ARGS←' '
[48]   ⍝ Select args, not fn name, unless to long:
[49]   ARGS←((LEN≤COLS)∧(N≥3 1 2)/1 0 1)≠ARGS
[50]   I←(HDR=';')/⍳LEN←⍴HDR ⍝ Indices of semicolon
[51]   N←⍴LEN←(1↓I,LEN)-I+1 ⍝ Length of localised name
[52]   MAX←⌈/LEN ⍝ Length of longest localised name
[53]   ⍝ Convert to matrix:
[54]   LOCALS←(N,COLS)↑(N,MAX)⍴(,LEN∘.>⍳MAX)\HDR←';'
[55]   LOCALS←¯1 0↓(LEN≤COLS)≠LOCALS ⍝ Ignore long ones (and extra one)
[56]   OK←=\CR≠'''' ⍝ Flag code NOT within quotes
[57]   ⍝ Flag code NOT within comment or quotes:
[58]   OK←OK∧∧\OK≤CR≠'⍝'
[59]   ⍝ Chars that can begin identifier-names:
[60]   LET←⎕AV[(65+⍳26),(97+⍳26)],'Δ_'
[61]   ⍝ Other valid chars:
[62]   NUM←'0123456789'
[63]   ⍝ Flag any valid chars, adding col for ending IDs:
[64]   VC←,(OK∧CR∈LET,NUM),0
[65]   ⍝ Flag start and end (+1) of each run:
[66]   VC←VC≠¯1↓0,VC
[67]   N←⍴IDS←VC/⍳⍴VC ⍝ Convert to indices
[68]   IDS←((N÷2),2)⍴IDS ⍝ Two col-mat of start/end indices
[69]   CR←,CR,' ' ⍝ Ravel code and pad to match VC
[70]   START←IDS[;0] ⍝ Start of indices
[71]   LEN←IDS[;1]-START ⍝ Length of names
[72]   ⍝ Must start with letter not be too long and not be preceded by '⎕':
[73]   ⍝
[74]   OK←(CR[START]∈LET)∧(LEN≤COLS)∧'⎕'≠CR[0⌈START-1]
[75]   START←OK/START ⍝ Squeeze out bad ones
[76]   N←⍴LEN←OK/LEN
[77]   IDS←(N×COLS)⍴' ' ⍝ IDS in raveled form
[78]   ⍝ I←(⍳LEN[0]),(⍳LEN[1]),(⍳LEN[2])...:
[79]   I←LEN/¯1↓0,+\LEN
[80]   I←(⍳⍴I)-I
[81]   ⍝ Insert IDS where they belong:
[82]   IDS[I+LEN/COLS×⍳N]←CR[I+LEN/START]
[83]   IDS←(N,COLS)⍴IDS ⍝ Reshape to a matrix
[84]   ⍝ Labels start a line and are folowed by a colon:
[85]   LABELS←((0=(WID+1)|START)∧CR[START+LEN]=':')≠IDS
[86]   IDS1←IDS
[87]   IDS←(0 0⍴⍨<\IDS∧.=⍉IDS)≠IDS ⍝ IDS without double items
[88]   IDS←SORTMAT IDS
[89]   ⍝ ANALYSE1: H.J.Veenendaal 18/03/00 11:29
       ∇
```

At Play With J: Blists in OLEIS

by Eugene McDonnell

This article discusses a kind of list I call a blist. The first part defines a blist, and covers material that is well known in combinatorial circles, and reported on by me in an earlier article [McD77], and also gives an actual use of J's Weighted Taylor Coefficients adverb `t:`. The second part breaks new ground, providing a tabulation that hasn't been seen before.

Part 1: What is a blist?

A *basic list*, or *blist*, is a list of length n with at least one of each of the items of `i. k`, where `1 <: k` and `k <: n`. For example, `0 2 1 0` is a blist, since it has at least one each of `i.3`, but `1 0 1 3` is not, since it has a three but no two. There is a many-to-one correspondence between the infinite number of arrays of n items and the corresponding finite number of blists of length n. The finitude of the number of blists of length n comes from the finitude of their permitted items. Since J's grade functions are omnivorous, the grade of any rank array can be found, and any array can be sorted, whether scalar or structured, boolean, integer, real, complex, literal, or boxed, and thus the blist of any array can be determined. The blist of an array can be determined by the function:

```
bl =: ] i.~ [: /:~ ~.
```

This finds the indices of the items of the array in the sorted nub of the array. For example, given the list:

```
   ] list =: ? 10 # 15
7 12 0 0 7 10 0 5 1 6
```

Its nub is

```
   ~.list
7 12 0 10 5 1 6
```

and its ordered nub is

```
   /:~~.list
0 1 5 6 7 10 12
```

and its indices in the ordered nub, that is, its blist, are

```
   List i.~ /:~ ~. List
4 6 0 0 4 5 0 2 1 3
```

and this has each of the values in `i.7` at least once.

A blist has the useful property that it has the same ordering relations as infinitely many other, more complex, lists and arrays, and thus can be substituted for those other lists and arrays in discussions of such properties. For example, an array and its blist have the same upgrade:

```
   List
7 12 0 0 7 10 0 5 1 6
   bL List
4 6 0 0 4 5 0 2 1 3
   /: List
2 3 6 8 7 9 0 4 5 1
   /: bL List
2 3 6 8 7 9 0 4 5 1
```

Other common properties of arrays and their blists are the same, for example their cycle structure, the number of operations needed to sort them, and their number of runs (up or down).

`BLT` is a brute-force function to give tables of all the blists of a given length. There is only one blist of length `1`, since the only permitted item is `0`.

```
   BLT 1
0
```

The blists of length two are:

```
   BLT 2
0 0
0 1
1 0
```

The blists of length three are:

```
   BLT 3
0 0 0
0 0 1
0 1 0
0 1 1
0 1 2
```

```
0 2 1
1 0 0
1 0 1
1 0 2
1 1 0
1 2 0
2 0 1
2 1 0
```

The number of blists of the first three orders can be counted easily: 1, 3, and 13. On the other hand, the function BLT soon runs out of space on my computer, requiring 4*n*n^n bytes to build the table from whose rows the blists are selected, and I can't use BLT beyond n=7. The space in bytes required for the tables for the first few values is given by:

```
   j=:13 : '4*y.*y.^y.'
   j 1 2 3 4 5 6
4 32 324 4096 62500 1119744
```

and for a few larger values:

```
   ,.j 7 8 9 10 11 12x
      23059204
     536870912
   13947137604
  400000000000
12553713506884
427972821516288
```

Although it is difficult to determine the values of blists of length n for large n, the number of such blists is much easier to find. The answer to exercise 5.3.1-3 in Knuth's *Searching and Sorting* volume gives a variety of ways for doing this. We can write a function F to give the number of blists of length n, based on Gross's formula $\Sigma_{k\geq 1} k^n/2^{1+k}$, $n \geq 1$. A version of Gross's formula in J is easy to write:

```
Gross =: 13 : '+/(x.^y.)%2^>:x.'
```

The formula implies an infinite number of values of k are required, but in practice I find that using the first 101 positive integers suffices.

```
k=: >:i.101
F =: k&Gross
```

The number of possible blists for arrays of ranks 1 through 15 are:

```
 n  F n      n          F n      n            F n
 1    1      6         4683     11     1622632573
 2    3      7        47293     12    28091567595
 3   13      8       545835     13   526858348381
 4   75      9      7087261     14 10641342970443
 5  541     10    102247563     15 230283190977853
```

The terminal digits of the values repeat in the pattern 1 3 3 5, so that if 4|n is 1 2 3 0, then 10|F n is 1 3 3 5, respectively, for positive n. This series is number A000670 in N. J. A. Sloane's magnificent website, *On-Line Encyclopedia of Integer Sequences* (OLEIS):

http://www. research. att. com/ ~njas/ sequences/

I shall be referring to Sloane's OLEIS numbers frequently in what follows.

This sequence of numbers arises naturally in a variety of areas, in addition to sorting, including trees with n+1 leaves, combination locks, compositions of numbers, and left arguments to APL's transpose dyad, but it is the sorting topic that is most interesting. It allows one to say in exactly how many ways an arbitrary list of length n can be arranged. For example, three items A, B, and C of any value can be arranged in thirteen ways, depending on their size interrelationships, using the relations = and < and the convention that A=B<C means (A=B)*.(B<C). Next to each relation list I've placed the corresponding blist, to show the kinship of the two forms.

```
A=B=C  0 0 0
A=B<C  0 0 1
A=C<B  0 1 0
A<B=C  0 1 1
A<B<C  0 1 2
A<C<B  0 2 1
B=C<A  1 0 0
B<A=C  1 0 1
B<A<C  1 0 2
C<A=B  1 1 0
C<A<B  1 2 0
B<C<A  2 0 1
C<B<A  2 1 0
```

Gross's formula, even after increasing the number of terms in the left argument, begins to lose accuracy after length 14. To obtain accurate values for P_n , the number of blists of length n, one can use the identity

$$2Pn=\Sigma_k\ (k!n) * P_{n-k}, \text{ for } n> 0$$

Instead of obtaining just one value, we obtain a list of all the values up to the one we're seeking, given that the first value is 1, and that all successive values can be appended to this value by a sum of the products of the list and a conforming list of binomials. For example, assuming we have the list 1 1 3, we can get the next two longer lists by:

```
   1 1 3 , +/ 1 1 3 * ((i.3)!3)
1 1 3 13
   1 1 3 13 , +/ 1 1 3 13 * ((i.4)!4)
1 1 3 13 75
```

The function LPA encapsulates this strategy:

```
   LPA =: 13 : 'y.,+/y.*(i.!])#y.'
   LPA 1
1 1
   LPA LPA 1
1 1 3
   LPA LPA LPA 1
1 1 3 13
   LPA^:3[1
1 1 3 13
```

To produce the list of the first n terms, one would write `LPA^:n 1`. Because the terms grow large quite rapidly, it is necessary to use extended arguments if terms of high degree are wanted. We can thus obtain arbitrarily large values.

```
   LPA =: 13 : 'y.,+/y.*(i.!])#y.'
   ,.28 29 30 31{LPA^:31[1x
     629756206495006603351837393533 4635
   263478385263023690020893329044576861
 114035687940118804837424641961849019 63
5100080365742693884308410240759181189 73
```

Approximate values can be obtained by the function L, built around the powers of the logarithm of 2 (this isn't in Knuth, I found it accidentally by playing with the series):

```
   L=: 13 : '(!y.)%+:(^.2)^>:y.'
   L 8
545834.99790748546
   L 9
7087261.0016229022
```

Rounding to the nearest integer, this function will give accurate results up to 13,

```
   <.0.5+L 13
526858348381
```

but is off by one for 14:

```
   <.0.5+L 14
106413429704 44
```

We know this can't be right, since if 4|14 is 2, then 10| L 14 must be 3, not 4.

Another way to get the number of blists for a given n is to use its exponential generating function (egf). I look back wistfully on myself at the age of 19 learning the calculus necessary to understand exponential generating functions, but in the 55 years since the knowledge has somehow departed from me. Now I can't tell you how to derive it, but will merely state it. If you have the necessary mathematical background to understand it (which I don't any more), you can read the answer to exercise 7.44 in Knuth, et al.'s *Concrete Mathematics*. In common mathematical notation the egf for the number of blists is $1/(2-e^n)$, and the J version can be written directly from this:

```
   paegf=: %@(2:-^)
```

The values don't seem to have much of a pattern:

```
   paegf i.11
1 _1.3922 _0.18556 _0.055293 _0.019012 _0.00683 _0.0024911
_0.00091355 _0.00033569 _0.00012344 _4.5404e_5
```

However, if you apply J's Weighted Taylor Coefficients adverb to it, it becomes a marvel:

```
  (paegf t:) i.11
1 1 3 13 75 541 4683 47293 545835 7087261 102247563
```

Lastly, and something else I discovered by playing with the series, if we divide the n-1th value by the nth, and multiply this by n, we get a number which more and more closely approaches the natural log of 2.

```
   qq=:LPA^:(20)1
   (1+i.20) * ,.2 %/\qq
                  1
0.66666666666666663
0.69230769230769229
```

```
0.69333333333333336
0.69316081330868762
0.69314541960281872
0.69314697735394248
0.69314719649710999
0.69314718337591907
0.69314718043695578
0.69314718052316582
0.69314718056053715
0.69314718056040159
0.69314718055994917
0.69314718055993996
0.69314718055994518
 0.6931471805599454
0.69314718055994529
0.69314718055994529
0.69314718055994529
```

Compare this with the machine-precision value of the logarithm of 2:

```
   ^. 2
0.69314718055994529
```

Part 2: How many blists of length n begin with k?

Inspecting the tables of blists of various lengths, I began to wonder how many of each table began with each possible number. I wrote this function to count how many times each leading digit appeared.

```
   CL =: [: #/.~ [: {."1 ]
```

I began building an upper triangle matrix (analogous to the Pascal triangle), where an entry in row i, column j gives the number of blists of length j+1 that begin with integer i.

```
   CL BLT 1
1
   CL BLT 2
2 1
   CL BLT 3
6 5 2
   CL BLT 4
26 25 18 6
   CL BLT 5
150 149 134 84 24
   CL BLT 6
1082 1081 1050 870 480 120
   CL BLT 7
9366 9365 9302 8700 6600 3240 720
```

The last one took a looong time. With these I was able to create table `t1`:

```
   t1
1 2 6 26 150 1082 9366
0 1 5 25 149 1081 9365
0 0 2 18 134 1050 9302
0 0 0  6  84  870 8700
0 0 0  0  24  480 6600
0 0 0  0   0  120 3240
0 0 0  0   0    0  720
```

Portions of this table appear in OLEIS. Its sum is A000670. The first row is Series A000629. The second row is one less than the first row, and is Series A002050. The third row is the first row minus 2^n, and is twice Series A002051. The lowest counterdiagonal is !n. The penultimate counterdiagonal is A038720. I massaged the numbers in various ways, the fruitful one being to take its first difference, providing a new last row of all zeros to preserve the data:

```
   ] t2=:2-/\t1,0
1 1 1  1  1   1    1
0 1 3  7 15  31   63
0 0 2 12 50 180  602
0 0 0  6 60 390 2100
0 0 0  0 24 360 3360
0 0 0  0  0 120 2520
0 0 0  0  0   0  720
```

This is series A028246 from OLEIS. It looks promising, but I want to remove the factorials from the rows:

```
   ] t3=:t2%!i.#t2
1 1 1 1  1  1   1
0 1 3 7 15 31  63
0 0 1 6 25 90 301
0 0 0 1 10 65 350
0 0 0 0  1 15 140
0 0 0 0  0  1  21
0 0 0 0  0  0   1
```

And this brings me to familiar territory. It is the table of the Stirling numbers of the second kind, also called *subset* numbers, and is series A008277 from OLEIS. It is usually displayed transposed from the table above, and with an added first row and first column.

```
   ((1+#t3){.1),.0,|:t3
1 0   0    0    0    0    0   0
0 1   0    0    0    0    0   0
0 1   1    0    0    0    0   0
0 1   3    1    0    0    0   0
0 1   7    6    1    0    0   0
0 1  15   25   10    1    0   0
0 1  31   90   65   15    1   0
0 1  63  301  350  140   21   1
```

The reason these are called subset numbers is that entry (i,j) gives the number of ways to partition a set of i items into j nonempty parts. Thus, the value 7 in row 4, column 2, says there are 7 ways to put 4 items into 2 nonempty parts:

```
   (abc,d);(abd,c);(acd,b);(bcd,a);(ab,cd);(ac,bd);(ad,bc)
```

```
      +/t3
1 2 5 15 52 203 877
```

These are the Bell numbers (series A000110), which give the total number of ways of placing n distinct objects in n boxes. This is to be expected, since the subset number in item (n;k) gives the number of ways to partition a set of n things into k nonempty subsets. The Bell numbers thus summarize the subset numbers.

But now I know how to create *my* table of numbers. I can use the nonrecursive function `s2nr` from Iverson's *Concrete Math Companion* to generate the subset numbers.

This may be a bit mysterious at first, so I'll show you how its parenthesized central portion works.

```
   s2nr=:|:@(^/~ %. ^!._1/~)@i."0
   ] v0 =. i.7x
0 1 2 3 4 5 6
```

Form the table of powers `t4`

```
   ] t4 =. ^/~ v0
1 0  0   0    0    0     0
1 1  1   1    1    1     1
1 2  4   8   16   32    64
1 3  9  27   81  243   729
1 4 16  64  256 1024  4096
1 5 25 125  625 3125 15625
1 6 36 216 1296 7776 46656
```

and the table of falling factorials `t5`

```
   ] t5 =. ^!._1/~ v0
1 0  0   0   0   0   0
1 1  0   0   0   0   0
1 2  2   0   0   0   0
1 3  6   6   0   0   0
1 4 12  24  24   0   0
1 5 20  60 120 120   0
1 6 30 120 360 720 720
```

and make these the left and right arguments to matrix divide, yielding the table of subset numbers.

```
   ] t6=: |: t4 %. t5
1 0  0  0  0  0 0
0 1  0  0  0  0 0
0 1  1  0  0  0 0
0 1  3  1  0  0 0
0 1  7  6  1  0 0
0 1 15 25 10  1 0
0 1 31 90 65 15 1
```

I now can produce the table of leading digits versus length of splits. The first step is to build a table of subset numbers.

```
   ] a =. s2nr 10x
1 0   0    0    0    0    0   0 0 0
0 1   0    0    0    0    0   0 0 0
0 1   1    0    0    0    0   0 0 0
0 1   3    1    0    0    0   0 0 0
0 1   7    6    1    0    0   0 0 0
0 1  15   25   10    1    0   0 0 0
0 1  31   90   65   15    1   0 0 0
0 1  63  301  350  140   21   1 0 0
0 1 127  966 1701 1050  266  28 1 0
0 1 255 3025 7770 6951 2646 462 36 1
```

Drop the leading row and column, then transpose.

```
   ] b =. |: 1 1 }. a
1 1 1 1  1  1   1    1    1
0 1 3 7 15 31  63  127  255
0 0 1 6 25 90 301  966 3025
0 0 0 1 10 65 350 1701 7770
0 0 0 0  1 15 140 1050 6951
0 0 0 0  0  1  21  266 2646
0 0 0 0  0  0   1   28  462
0 0 0 0  0  0   0    1   36
0 0 0 0  0  0   0    0    1
```

Multiply row i by factorial i.

```
   ] c =. -b * ! i. # b
1 1 1   1   1    1     1      1       1
0 1 3   7  15   31    63    127     255
0 0 2  12  50  180   602   1932    6050
0 0 0   6  60  390  2100  10206   46620
0 0 0   0  24  360  3360  25200  166824
0 0 0   0   0  120  2520  31920  317520
0 0 0   0   0    0   720  20160  332640
0 0 0   0   0    0     0   5040  181440
0 0 0   0   0    0     0      0   40320
```

Sum from the bottom up.

```
   ] d =. +/ \. c
1 2 6 26 150 1082 9366 94586 1091670
0 1 5 25 149 1081 9365 94585 1091669
0 0 2 18 134 1050 9302 94458 1091414
0 0 0  6  84  870 8700 92526 1085364
0 0 0  0  24  480 6600 82320 1038744
0 0 0  0   0  120 3240 57120  871920
0 0 0  0   0    0  720 25200  554400
0 0 0  0   0    0    0  5040  221760
0 0 0  0   0    0    0     0   40320
```

And this is the table I wanted to be able to create.

I've told you the series numbers in OLEIS of parts of my table. What about the table itself? I'm pleased and proud to be able to tell you that when I emailed information about it to Neil Sloane, proprietor of OLEIS, he agreed it was new, and assigned the number A054255 to it, with credit to me. I feel as if I've gained a speck of immortality. We now have blists in OLEIS. You can look it up!

Reference

[McD77] McDonnell, Eugene; "How Shall I Transpose Thee? Let Me Count the Ways", *APL Quote Quad*, vol. 8, no. 1, September 1977

OOF: Getting a better Grip on OLE Objects

by Morten Kromberg (mkrom@insight.dk)

This paper was first published in FinnAPL News.

Introduction

During the last few years, I have made a few short experimental trips into Object Land, primarily from Dyalog APL, and each time I ended up feeling intensely frustrated. I seemed to have to go through endless steps in order to access functionality which, judging from code examples in magazines, seemed to be much more accessible to users of "inferior" tools like Visual Basic. All the code examples I found in magazines, on the web, or using the Excel Macro Recorder, required a real translation effort, and turned into somewhere between three and ten times as many lines of code as the original. This article describes the results of my attempts to improve this situation; a tool which is freely available to anyone who is interested (see under Acknowledgements below).

For several years, I successfully evaded the issue by assuming the guise of a manager (apologies to those of you who are not familiar with this term; try looking it up on the web). Finding myself unable to keep this up indefinitely, I have now accepted that it is inevitable that I must try to understand objects a bit better. Being an APL'er, the next step is obviously to build a tool which maps the world into my preferred co-ordinate system. I named the tool "OOF" because, in addition to being a TLA (Three Letter Acronym) for OLE Object Functions, "OOF" is "FOO" backwards, and symbolises the struggle of an ageing developer who wants to get rid of all those statements which simply manipulate objects, and get back to the function calls which he understands.

The OLE Server that I needed to use is one of the most common: Microsoft Excel. As an example, if the task is to retrieve the values in the range [A1;D3] from the first sheet in a particular book, the "raw" Dyalog APL code required to do this is:

```
      'XL' ⎕WC 'OLEClient' 'Excel.Application'
      'WBS' ⎕WC 'XL' ⎕WG 'WorkBooks'
      'BOOK' ⎕WC 3 ⎕NQ 'WBS' 'Open' 'MyBook.xls'
      'SHEETS' ⎕WC 'BOOK' ⎕WG 'WorkSheets'
      'SHEET1' ⎕WC 'SHEETS' ⎕WG 'Item[1]'
      'RANGE' ⎕WC 'SHEET1' ⎕WG 'Range[A1;D3]'
      ⍴'RANGE' ⎕WG 'Value'
3 4
```

With OOF, the corresponding code is:

```
      #.OOF.Add '#.XL' 'Excel.Application'
      #.OOF.Add '#.BOOK' '#.XL.Workbooks.Open' 'MyBook.xls'
      ⍴#.OOF.Prop '#.BOOK.Sheets[Sheet1].Range[A1;D3].Value'
3 4
```

Acknowledgements

Many thanks to Carlo Alberto Spinicci of APL Italiana, who paid for the work described in this article, and graciously agreed not only to allow me to publish this article, but also to make the tool available to anyone who is interested. If you would like a copy, write to oof@insight.dk. I can heartily recommend this approach: save printing costs and get an APL publication to print your internal documentation for you!

Thanks also to Peter Donnelly and John Daintree of Dyadic Systems Ltd. for patiently answering stupid questions about whether "Item" is a property or a method.

A Map of The Battle Field

Any OLE Server exposes an Object Model, which you must understand in order to use it. Microsoft Excel has a huge and complicated (sorry, "provides a rich") hierarchy of objects. The following picture illustrates my understanding of the very tiny fragment of the Excel Object model which we "navigated" in the introductory example:

At the top of the hierarchy is the Excel.Application, the object to which we connect when we create our OLE Client object. Each object has properties and methods. A property can be simple, like the Name property which contains the string "Microsoft Excel". Alternatively, the property can be another object, with its own properties and methods. Methods can have no result, simple results, or return objects. For example, the "Range" method of a worksheet returns an object which contains the cells in the range.

In the case of Excel, the operations that we would like to perform on the objects are things like:

- Open our favourite book containing spreadsheets:
  `WorkBooks.Open "MyBook"`
- Query or set the value of a range
  `WorkBooks[MyBook].WorkSheets[Sheet1].Range[A1;D3].Value`

- Execute methods, like
  `WorkBooks[MyBook].Save`, or
  `WorkBooks[MyBook].PrintOut`

Introduction to the Dyalog APL OLEClient

Before talking more about OOF, let's take a look at the native mechanisms provided by Dyalog APL for manipulation of object models:

Connecting to an OLE Server

Or, seen from the APL side, creating an OLE Client:

```
'XL' ⎕WC 'OLEClient' 'Excel.Application'
```

This creates a namespace which represents the object in the APL workspace.

Accessing Properties and Methods

Dyalog APL provides no less than 3 mechnisms(!) for accessing properties and methods, once you have created a namespace representing an object:

- By setting the `'AutoBrowse'` property of a new object to `1`, you can get APL to scan libraries and immediately define all properties as APL variables and all methods as APL functions within the new namespace:

```
      'XL' ⎕WC 'OLEClient' 'Excel.Application' ('AutoBrowse' 1)
      XL.Name
Microsoft Excel
      XL.InchesToPoints 1
72
```

Unfortunately, this is too slow for runtime applications, our original example could easily take 30 seconds to execute using this technique.

- `2 ⎕NQ 'SetPropertyInfo'` and `'SetMethodInfo'` allow you to control the creation of individual properties or methods:

```
      'XL' ⎕WC 'OLEClient' 'Excel.Application'
      2 ⎕NQ 'XL' 'SetPropertyInfo' 'Name'
      2 ⎕NQ 'XL' 'SetMethodInfo' 'InchesToPoints'
      XL.Name (XL.InchesToPoints 1)
Microsoft Excel  72
```

This runs quite fast, but adds one statement for each property or method you wish to use.

- Finally, ⎕WG and 3 ⎕NQ give direct access to properties and methods without 'creating' them first:

```
      '#.XL' ⎕WC 'OLEClient' 'Excel.Application'
      '#.XL' ⎕WG 'Name'
Microsoft Excel
    3 ⎕NQ '#.XL' 'InchesToPoints' 1
72
```

Building the Hierarchy in your Workspace

If a property or method returns an object, ⎕WC is used to create namespace for next level:

```
'XL' ⎕WC 'OLEClient' 'Excel.Application'
'WBS' ⎕WC 'XL' ⎕WG 'WorkBooks'
'BOOK' ⎕WC 3 ⎕NQ 'WBS' 'Open' 'MyBook.xls'
```

This allows you to reach your way through the hierarchy, creating a new namespace for each object you need to pass through. You can choose whether to create all these namespaces at the same level, as in the above example, or reflect the hierarchy by creating the objects inside their parent (this makes it easier to tidy up: simply delete the object at the top):

```
'XL' ⎕WC 'OLEClient' 'Excel.Application'
'XL.WBS' ⎕WC 'XL' ⎕WG 'WorkBooks'
'XL.WBS.BOOK' ⎕WC 3 ⎕NQ 'XL.WBS' 'Open' 'MyBook.xls'
```

Just be careful that you do not use names which conflict with any properties or methods at each level.

Collections

In object oriented terminology, a collection is an object which has a Count property and an Item property or method:

```
      '#.XL' ⎕WC 'OLEClient' 'Excel.Application'
      '#.WBS' ⎕WC '#.XL' ⎕WG 'WorkBooks'
      '#.WBS' ⎕WG 'Count'
0
      z←3 ⎕NQ '#.WBS' 'Open' 'MyBook.xls'
      '#.WBS' ⎕WG 'Count'
1
      '#.BOOK' ⎕WC '#.WBS' ⎕WG 'Item[1]'
```

Note that we used ⎕WG to read the Item as a property. In environments like Visual Basic, the distinction between a "parameterised property" and a method is blurred (at least with my eyesight). This can cause quite a bit of grief, since you are never sure which it is. OOF contains nasty trap statements to try first one then the other, to hide this – just how well this works in practice remains to be seen.

If I understood it all correctly, a collection will also have Item defined as its DEFAULT method, so that a Visual Basic expression like Workbooks[1] is equivalent to calling Workbooks.Item with an argument of 1. Many Item functions also support string arguments, allowing indexing by name.

Doing Something

We now know enough to understand what the example code in the introduction does. It creates 6 objects, corresponding to successively deeper levels of hierarchy (Excel-WorkBooks-Book1-Worksheets-Sheet1-Range). Finally, we can access the Value property of the range, which the Visual Basic developer might refer to as *WorkBooks[MyBook].WorkSheets[Sheet1].Range[A1;D3].Value*:

```
      'XL' ⎕WC 'OLEClient' 'Excel.Application'
     'WBS' ⎕WC 'XL' ⎕WG 'WorkBooks'
     'BOOK' ⎕WC 3 ⎕NQ 'WBS' 'Open' 'MyBook.xls'
     'SHEETS' ⎕WC 'BOOK' ⎕WG 'WorkSheets'
     'SHEET1' ⎕WC 'SHEETS' ⎕WG 'Item[1]'
     'RANGE' ⎕WC 'SHEET1' ⎕WG 'Range[A1;D3]'
     ⍴'RANGE'⎕WG'Value'
3 4
     'RANGE' ⎕WS 'Value' (3 4⍴⍳12)
```

In my opinion, it is a bit hard to see the forest for the trees.

Introducing OOF

With OOF, APL expressions correspond much more closely to the expressions which a Visual Basic developer would be able to use. The above example could be written as:

```
     #.OOF.Add '#.XL' 'Excel.Application'
     #.OOF.Add '#.BOOK' '#.XL.Workbooks.Open' 'MyBook.xls'
     ⍴#.OOF.Prop '#.BOOK.Sheets[Sheet1].Range[A1;D3].Value'
3 4
     (3 4⍴⍳12) #.OOF.Prop
'#.BOOK.Sheets[Sheet1].Range[A1;D3].Value'
```

The big difference is that you do not need to explicitly create all the intermediate objects required to reach into the hierarchy. OOF does this for you, exactly as Visual Basic does for its users. The function OOF.Objects shows the objects which OOF is managing for you:

```
      #.OOF.Objects
#.XL                                  #.XL
#.XL.Workbooks                        X0001
#.BOOK                                #.BOOK
#.BOOK.Sheets                         X0002
#.BOOK.Sheets[Sheet1]                 X0003
#.BOOK.Sheets[Sheet1].Range[A1;D3]    X0004
```

In other words, OOF has created Dyalog APL namespaces `X0001-X0004` because these intermediate namespaces were required in order to reach into the hierarchy as requested. The two objects #.XL and #.BOOK were named by us, using the function `#OOF.Add`.

Internally Visual Basic does the same thing; but it immediately disposes of the temporary objects. Apparently, many VB developers write quite ineffective code because they do not realise that they are causing an enormous amount of creation and garbage collection of objects, and that they would be better off explicitly assigning frequently-used objects to named variables. OOF keeps objects until you explicitly dispose of them using `OOF.Close` (you only have to close objects you created yourself, when you do, OOF will close any related objects it created on your behalf).

OOF consists of about half a dozen functions:

#.OOF.Add

Adds a new object in one of three ways:

- Attach to new OLE Server (second element or argument does not start with #, and contains a single ".")

```
#.OOF.Add '#.XL' 'Excel.Application'
```

- Create an object from property of existing object (second element does start with #)

```
#.OOF.Add '#.WBS' '#.XL.WorkBooks'
```

- Create an object from the result of Method (argument has more than 2 elements)

```
#.OOF.Add '#.BOOK' '#.XL.Workbooks.Add' 0
```

#.OOF.Prop

Queries or sets a property of an existing object, or an object which can be reached from an existing object by extending the hierarchy:

- Monadic calls query a property:

```
      #.OOF.Prop '#.XL.OperatingSystem'
Windows (32-bit) NT 5.00
```

- Dyadic calls set a property. The previous setting is always returned:

```
      1 #.OOF.Prop '#.XL.Visible'
0
```

Note that Prop also works with array properties like the Value of a Range.

#.OOF.Invoke and InvokeNil

Invoke a method in an existing object, or an object which can be reached from an existing object. If the method does not return a result, you must use the function *InvokeNil*. Examples:

- Get Excel to multiply by 72 for you:

```
      #.OOF.Invoke '#.XL.InchesToPoints' 1
72
```

Close the workbook without saving it (set Saved property to avoid Excel asking the user whether to save):

```
      1 #.OOF.Prop '#.BOOK.Saved'
      #.OOF.InvokeNil '#.BOOK.Close' 0
```

If you use *Invoke* instead of *InvokeNil*, you will get a value error. Note that, if a method takes several arguments, the third element of the argument must be a list of all of the arguments. Do NOT use elements following the third element for additional arguments.

#.OOF.CollProp

Being APL developers, we must have a way to manipulate the properties of multiple items of a collection in a single call! OOF.CollProp is a user-defined operator which allows us to:

- Read value of one or more properties (right operator argument) from one or more items (left operator argument):

```
      (1 #.OOF.CollProp 'Name')'#.BOOK.Sheets '
Sheet1
```

- Set the value(s) – returns old value(s):

```
      'Demo'(1 #.OOF.CollProp 'Name')'#.BOOK.Sheets'
Sheet1
```

- Query or set property or properties for all items:

```
      ('*' OOF.CollProp 'Name') '#.BOOK.Sheets'
Demo  Sheet2  Sheet3
```

Note that the left operator argument can have any value accepted by the Item function of the collection, we could have used `'Sheet1'` as an index in the second example. `'*'` is special-cased, using the function `OOF.GetIndexSet` (see below).

#.OOF.GetIndexSet

The "index set" of a collection is not always a dense set of integers from 1 to the value of the Count property of the collection. The function `GetIndexSet` returns the full index set of a collection:

```
      #.OOF.GetIndexSet '#.BOOK.Sheets'
1 2 3
```

Is the collection is indexed by an *enumerated type*, `GetIndexSet` returns a vector of enclosed two-element vectors, containing the names and values of the constants defining the set:

```
      +#.OOF.GetIndexSet '#.RANGE.Borders'
xlInsideHorizontal  12
xlInsideVertical    11
xlDiagonalDown       5
xlDiagonalUp         6
xlEdgeBottom         9
xlEdgeLeft           7
xlEdgeRight         10
xlEdgeTop            8
```

If you don't want to check the depth of the result, use a left argument of 1 to state that you want numeric indices only:

```
      #.OOF.GetIndexSet '#.RANGE.Borders'
12 11 5 6 9 7 10 8
```

#.OOF.Close

Close an object, plus all associated objects which OOF has created on your behalf.

```
      #.OOF.Objects
#.XL               #.XL
#.BOOK             #.BOOK
#.SHEET            #.SHEET
#.BOOK.Sheets      X0001
#.BOOK.Sheets[1]   X0002

    #.OOF.Close '#.BOOK'
    #.OOF.Objects
#.XL     #.XL
#.SHEET  #.SHEET
```

You can close associated objects at any time, if you know that you will not need them again. For example, we could have made the call:

```
    #.OOF.Close '#.BOOK.Sheets'
```

If we knew that we did not want to work on sheets any more. This would have expunged X0001 and X0002, since X0002 was a child of X0001, but left #.BOOK intact. Future versions of OOF may incorporate some kind of automatic garbage-collection mechanism (suggestions welcome)!

#.OOF.Investigate (experimental)

This is an experimental function which, if applied to an object which has 'AutoBrowse' set to 1, "tastes" all the properties and attempts to let you know something about them.

```
      #.OOF.AutoBrowse←1 ⍝ Applies to all OOF-created objects
      #.OOF.Add '#.XL' 'Excel.Application'
      #.OOF.Investigate '#.XL'
_Default                          Microsoft Excel  0  Normal
ActiveCell                                         2  Object
ActiveChart                                  0     0  Normal
ActiveDialog                                 0     0  Normal
ActiveMenuBar                                      2  Object
ActivePrinter  \\SRVLAN\HP4050 PCL 6 on Ne00:       0  Normal
ActiveSheet                                        2  Object

... etc ...
```

Conclusion

Dyalog APL provides most of what you need to use OLE Servers from APL, but in my opinion, they could be significantly easier to use. The "OOF" tool allows APL developers to use expressions which are closer to that which they can find in documentation and other literature, which is usually aimed at Visual Basic or Visual C developers.

However, I can't help feeling that I should not have had to build this tool - it should have been part of my APL system! It will soon be time to re-launch APL to a new generation of developers. The people with an "attitude" to APL (good or bad) are moving on, and I think we will soon have an opportunity to try to sell APL again. This will not be possible if the new audience feel that the mechanisms we provide for inter-application communication are significantly inferior to the environments they are used to. On the other hand - if we have it, there is no reason why we should not be able to go out and sell APL to people who need to compute something, all the other languages are still far behind APL in this respect.

If the vendors are not already working on it, this seems to be the next big challenge which we need to address as a community. I think I would go as far as to say that the long term survival of APL depends on support for objects at the "language" level.

Bell Numbers and APL

by Joseph De Kerf

Abstract

In 1934 E.T. Bell introduced what he called the exponential numbers. Defined functions for calculating those numbers, as well as for calculating the rows of the so-called Bell triangles, are given. Results of benchmarks, comparing the efficiency in cpu time, are reported. The relation with the Stirling numbers of the second kind is treated in an Appendix.

Bell Numbers

In 1934 E.T. Bell [1], [2], author of several detective stories under the name John Taine, introduced what he called the exponential numbers B_0, B_1, B_2, ... as those defined by the Maclaurin expansion of the generating function $(e*e*x) \div e$:

$$e^{e^x} = e\sum_{n=0}^{\infty} B_n \frac{x^n}{!n}$$

$$= e\left(B_0 + B_1\frac{x}{!1} + B_2\frac{x^2}{!2} + B_3\frac{x^3}{!3} + \ldots\right)$$

with
$B_0 = 1$
$B_1 = 1$
$B_2 = 2$
$B_3 = 5$
$B_4 = 15$
$B_5 = 52$
$B_6 = 203$
$B_7 = 877$
$B_8 = 4140$
$B_9 = 21147$
$B_{10} = 115975$
.....

Bell showed that those numbers may be calculated by binomial expansion of the form $B_n = (1+B)^{n-1}$ and substituting B^i by B_i, giving the recursion formula:

$$B_0 = 1$$

.....

$$B_n = \sum_{i=0}^{n-1} \binom{n-1}{i} B_i$$

Bell also showed that the numbers may be calculated explicitly by the form (n>0):

$$B_n = \sum_{s=1}^{n} \frac{1}{!(s-1)} \left[\sum_{r=0}^{s-1} (-1)^r \binom{s-1}{r} (s-r)^{n-1} \right]$$

Finally, L.F. Epstein [3] showed that the Bell numbers may be defined by the infinite series:

$$B_n = \frac{1}{e} \sum_{k=0}^{\infty} \frac{k^n}{!k}$$

as such extending the domain to real and complex numbers. He also gave several asymptotic formulas, as well as the values of B_n for n = 1(1)20.

More elaborate tables were published for n = 0(1)50 by H. Gupta [4], and for n = 0(1)74 by J. Levine and R.E. Dalton [5].

Bell numbers are rather unknown, even in the mathematical community. In most books on number theory and numeric analysis they are not mentioned, and this notwithstanding their practical importance. For instance, B_n specifies the number of rhyming schemes in the stanza of n lines; the number of pattern sequences for words of n letters, as used in cryptology; the number of trays n unlike objects can be placed in 1(1)n like boxes, allowing blank boxes; and the number of ways a product of n distinct primes may be factorised. References are given in [5].

The purpose of this paper is to give some defined functions for calculating those Bell numbers, as well as for calculating the rows of the so-called Bell triangle. Results of benchmarks comparing the efficiency in cpu time are reported. The relation with the Stirling set numbers of the second kind is treated in an Appendix.

Accuracy of the software is approximately 15 digits throughout. It should be noticed that, if print precision is set to 15, for $n \leq 21$ the numbers are displayed as

integers, while for $n \geq 22$ the numbers are rounded off and displayed in exponential notation:

B_{21} = 474869816156751

B_{22} = 4.50671573844732E15

Defined functions shown conform to the proposed new standard ISO APL-Extended, prepared by the ISO APL Working Group ISO-IEC/JTC1/SC22/WG3 [8]. Benchmarks have been done on a MicroLine Pentium S-100, with mathematical co-processor, using Dyalog APL/W Version 7.1.2 under Windows 3.11 [9]. Migration Level ⎕*ML* was set to 2, the default value being 0. CPU times were measured, using the system function ⎕*MONITOR*.

Defined Functions

APL defined functions *BELL1* and *BELL2*, returning as explicit result the Bell sequence $B_0, B_1, \ldots, B_n$ and based on the recursion formula

$$B_0 = 1$$

.....

$$B_n = \sum_{i=0}^{n-1} \binom{n-1}{i} B_i$$

$$= \binom{n-1}{0} B_0 + \binom{n-1}{1} B_1 + \ldots + \binom{n-1}{n-1} B_{n-1}$$

are for instance:

```
      ∇Z←BELL1 N
[1]     →(N=0)/0,Z←,1
[2]     Z←BELL1 N-1
[3]     Z←Z,+/Z×(¯1+⍳⍴Z)!¯1+⍴Z
      ∇

      BELL1 10
1 1 2 5 15 52 203 877 4140 21147 115975

      ∇Z←BELL2 N
[1]     →(N=0)/0,Z←,1
[2]     END:Z←Z,+/Z×(¯1+⍳⍴Z)!¯1+⍴Z
[3]     →(N≥⍴Z)/END
      ∇

      BELL2 10
1 1 2 5 15 52 203 877 4140 21147 115975
```

*BELL*1 uses implicit recursion. *BELL*2 uses a programmed recursion loop. The maximum of the domain of the right argument N is N = 218 as B_{218} = 6.10130983387532E308 and B_{219} exceeds the largest number representable 1.79769313486232E308.

Recursion may be avoided by direct calculation of the numbers, for instance through appropriate truncation of the infinite series of Epstein:

$$B_n = \frac{1}{e}\sum_{k=0}^{\infty}\frac{k^n}{!k}$$
$$= \frac{1}{e}\left(\frac{0^n}{!0}+\frac{1^n}{!1}+\frac{2^n}{!2}+\cdots\right)$$

Successive partial sums may be calculated and compared, the process being ended when two successive partial sums are equal within the accuracy desired (15 digits). However, this needs a loop.

The defined function *BELL*3 avoids the use of such a loop by calculating *a priori* the number K of terms needed to achieve the desired accuracy (15 digits):

```
      ∇Z←BELL3 N;K
[1]    →(N=0)/0,Z←,1
[2]    END:K←(21≥ρZ)×⌊0.48+1.36×ρZ
[3]    K←K+(22≤ρZ)×⌊21.83+(0.407-0.000616×ρZ)×ρZ
[4]    →(N≥ρZ←Z,⌈(+/(K*ρZ)÷!K←⍳K)÷*1)/END
      ∇

       BELL3 10
1 1 2 5 15 52 203 877 4140 21147 115975
```

The ceiling of the partial sum is taken such that for N = 0 only 1 term instead of 18 terms is needed, for N = 1 only 1 term instead of 19 terms is needed, for N = 2 only 2 terms instead of 19 terms are needed, etc. Minima K_m of K in the function of N were determined. Using the method of least squares, a straight line was drawn through those minima for the domain $n \le 21$ and a second degree polynom for the domain $n \ge 22$.

Finally the parameters were adjusted so that the calculated K_c is greater than or equal to K_m and the overhead $\Delta = K_c - K_m$ is minimised, from 0 to 2 terms maximum:

$n \le 21$: $K_c = \lfloor 1.48+1.36\times N$
$n \ge 22$: $K_c = \lfloor 22.83+(0.407-0.000616\times N)\times N$

Values of K_m and K_c, as well as the overhead $\Delta = K_c - K_m$, are listed for $N = 10(10)160$ in the table below:

N	K_m	K_c	Δ	N	K_m	K_c	Δ
10	15	15	0	90	52	54	2
20	28	28	0	100	56	57	1
30	34	34	0	110	58	60	2
40	36	38	2	120	61	62	1
50	39	41	2	130	63	65	2
60	44	45	1	140	67	67	0
70	46	48	2	150	70	70	0
80	50	51	1	160	71	72	1

Finally, K is set to $K_c - 1$, as for $N>0$ the first term of the series must not be calculated, this term being 0 ($0*N = 0$ and $!0 = 1$).

It should be noticed that the maximum of the domain of the right argument N is $N = 165$ instead of $N = 218$, with $B_{165} = 1.5641005513217E217$, as for $N = 166$ the minimum K_m is 73 and 72*166 exceeds the largest number representable 1.79769313486232E308. If $N>165$, an error report is generated.

In most applications, the Bell numbers are only requested for a selected set of the sequence $B_0, B_1, \ldots, B_n$. A modified form *BELL*4 of the defined function *BELL*3, that only returns the scalar B_N is given below:

```
      ∇Z←BELL4 N;K
[1]     K←(N≤21)×⌊1.48+1.36×N
[2]     K←K+(N≥22)×⌊22.83+(0.407-0.000616×N)×N
[3]     Z←⊃(+/(K*N)÷!K←¯1+⍳K)÷*1
      ∇

      BELL4 0
1

      BELL4 10
115975

      BELL4¨ 0,⍳10
1 1 2 5 15 52 203 877 4140 21147 115975
```

Of course, just as for *BELL*3, the maximum of the domain of the right argument N is $N = 165$, instead of $N = 218$. If $N>165$, an error report is generated.

Note: In some implementations, as N increases, the accuracy of the results decreases from 15 to 14 digits. This is due, in those implementations, to the lack of accuracy of the monadic function factorial !R for higher values of the argument R. The discrepancy may be removed by substituting:

in *BELL*3 the expression `!K←⍳K`
by `×\K←⍳K`

in *BELL*4 the expression `!K←¯1+⍳K`
by `×\1,1+K←¯1+⍳K`

Benchmarks

Benchmarks on cpu time have been done for N = 40(40)200. Results are shown below (in ms):

N	40	80	120	160	200
*BELL*1 *N*	29	106	270	557	999
*BELL*2 *N*	25	100	261	545	984
*BELL*3 *N*	42	102	180	271	-
*BELL*4 *N*	2.01	2.40	2.75	3.06	-
*BELL*4`¨ 0,⍳N`	42	103	181	272	-

As N increases, *BELL*2 is from 16.0% to 1.5% more efficient in cpu time than *BELL*1, the difference becoming negligible for higher values of N. For lower values of N, *BELL*3 is about 56% slower than *BELL*1 and *BELL*2, but 2.0 times more efficient for N = 160, the breakpoint being at about N = 80. Finally, as expected, the overhead usoing *BELL*4 instead of *BELL*3 for generating the complete sequence $B_0, B_1, \ldots, B_n$ is negligible.

The Bell Triangle

In 1980, J. Shallit [6] defined what he called the Bell Triangle $T(n,k)$ with $k = 1(1)n$, as generated by the recursion algorithm:

$$T(1,1) = 1$$
$$T(n,1) = T(n-1,n-1) \qquad (2 \le n)$$
$$T(n,k) = T(n,k-1) + T(n-1,k-1) \qquad (2 \le k \le n)$$

The inconvenience of this algorithm is that, for calculating $T(n,k)$, the predecessor $T(n,k-1)$ in its row must be known. This inconvenience may be removed by transforming the algorithmn to the form:

$$T(1,1) = 1$$
$$T(n,1) = T(n-1,n-1) \qquad (2 \le n)$$
$$T(n,k) = T(n-1,n-1) + \sum_{r=1}^{k-1} T(n-1,r) \qquad (2 \le k \le n)$$

such that the elements of the nth row $T(n,k)$ may be calculated from the elements of the previous row only.

The defined function *BELLTR1*, based on the algorithm in its transformed version, returns as explicit result a nested vector of vectors containing the N first rows of the triangle:

```
      ∇Z←BELLTR1 N
[1]    →(N=1)/0,Z←1ρ⊂,1
[2]    Z←BELLTR1 N-1
[3]    Z←Z,⊂(↑⌽↑⌽Z)+0,+\⊃↑⌽Z
      ∇
```

The expression `⊃⍪⍤¯1⊢BELLTR1 N` transforms the nested vector of vectors generated by *BELLTR1 N* to a simple character matrix containing the N rows of the triangle, adjusted to the left, and padded with blanks up to the character length of the last row. The defined function *CENTER* rotates the rows of this character matrix such that those rows are centred. The function *CENTER* has been taken from the book *APL2 in Depth* by N.D. Thomson and R.P. Polivka [7].

```
      ∇Z←CENTER M;R
[1]    R←+/∧\' '=⌽M
[2]    Z←(-⌊0.5×R)⌽M
      ∇
```

```
      CENTER ⊃⍕¨BELLTR1 8
                    1
                  1 2
                2 3 5
              5 7 10 15
           15 20 27 37 52
        52 67 87 114 151 203
     203 255 322 409 523 674 877
877 1080 1335 1657 2066 2589 3263 4140
```

The nth row contains n elements. Rows are not symmetric. The left side of the triangle is the Bell sequence $B_0, B_1, B_2, \ldots, B_0$ included, and the right side of the triangle is the Bell sequence $B_1, B_2, B_3, \ldots$. This means that the algorithm may be used to generate the Bell sequence:

$$B_0, B_1, \ldots, B_N = \text{⍎¨BELLTR1 } N+1$$
$$= 1, \text{⍎¨⍕¨BELLTR1 } N$$

and this for up to $N = 217$ and 218 respectively. Indeed, the 219th row starts with B_{218} but ends with B_{219}, and this exceeds the largest number representable 1.79769313486232E308. Examples:

```
      ⍎¨BELLTR1 10+1
1 1 2 5 15 52 203 877 4140 21147 115975

      1,⍎¨⍕¨BELLTR1 10
1 1 2 5 15 52 203 877 4140 21147 115975
```

Shallit reported several properties of the elements of the triangle, for instance that the sum of the elements of the nth row is $B_{n+1} - B_n$:

$$\sum_{k=1}^{n} T(n,k) = T(n,1) + T(n,2) + \cdots + T(n,n)$$
$$= B_{n+1} - B_n$$

Examples:

```
      +/¨BELLTR1 10
1 3 10 37 151 674 3263 17007 94828 562595

      (2↓BELL3 11)-1↓BELL3 10
1 3 10 37 151 674 3263 17007 94828 562595
```

BELLTR1 uses implicit recursion. A defined function *BELLTR2*, based on the same algorithm, but using a programmed recursion loop, may be:

```
∇Z←BELLTR2 N
 →(N=1)/0,Z←1⍴⊂,1
 END:Z←Z,⊂(↑⌽↑⌽Z)+0,+\⊃↑⌽Z
 →(N>⍴Z)/END
∇
```

Bell Numbers and the Bell Triangle

The question arises, if generating the Bell sequence by using the defined functions *BELLTR1* or *BELLTR2* may compete in cpu time with the direct calculation from *BELL1*/*BELL2* or *BELL3*. Benchmarks have been done for N = 40(40)200.

Results are shown below (in ms):

N	40	80	120	160	200
↑¨*BELLTR1 N*+1	28	62	114	183	269
↑¨*BELLTR2 N*+1	20	47	92	154	233
1,↑¨⌽¨*BELLTR1 N*	28	65	121	195	289
1,↑¨⌽¨*BELLTR2 N*	20	50	99	166	251

As N increases, *BELLTR2* is from 40% to about 15.3% more efficient in cpu time than *BELLTR1*. The second form is about up to 7.6% slower than the first form, but, as already mentioned, it works for up to N = 218 instead of N = 217. Most important, however, is that the cpu times reported are drastically lower than those for *BELL1*/*BELL2* and *BELL3*.

In addition, the process may be considerably accelerated by incorporating the deduction of the Bell numbers from the rows of the Bell triangle in the defined functions themselves, such as is done in the defined functions *BELLA* and *BELLB*:

```
      ∇Z←BELLA N
[1]    →(N≤1)/0,Z←(1+N⌊1)⍴R←1
[2]    Z←BELLA N-1
[3]    Z←Z,¯1↑R←(¯1↑R)+0,+\R
      ∇
```

```
      BELLA 10
1 1 2 5 15 52 203 877 4140 21147 115975

     ∇Z←BELLB N;R
[1]   →(N≤1)/0,Z←(1+N⌊1)ρR←1
[2]   END:Z←Z,¯1↑R←(¯1↑R)+0,+\R
[3]   →(N≥ρZ)/END
     ∇

      BELLB 10
1 1 2 5 15 52 203 877 4140 21147 115975
```

Both functions cover the domain $N = 0(1)218$. *BELLA* uses implicit recursion. *BELLB* uses a programmed recursion loop. Benchmarks have been done for $N = 40(40)200$. Results are shown below (in ms):

N	40	80	120	160	200
BELLA N	18	39	62	89	118
BELLB N	13	28	46	67	91

As N increases, *BELLB* is from 38.5% to 29.7% more efficient in cpu time than *BELLA*. As N increases, *BELLB* is from 3.2% to 4.0% more efficient in cpu time than *BELL3* and from 2.1 to 10.9 times more efficient than *BELL1* and *BELL2*. Apparently, this substantial improvement is mainly due to the fact the algorithm only uses addition. It is doubtful if an algorithm that performs considerably better can be found.

References

[1] E.T. Bell: *Exponential Numbers*; American Monthly, Vol.41, 1934, pp. 411–419

[2] E.T.Bell: *The Iterated Exponential Integers*; Annals of Mathematics, Vol.39, 1938, pp. 539–557

[3] L.F. Epstein: *A Function Related to the Series e*e*x*; Journal of Mathematics and Physics, Vol.18, 1939, pp. 153–173

[4] H. Gupta: *Tables of Distribution*; East Punjab University Research Bulletin, Vol.2, 1950, p.44

[5] J. Levine & R.E. Dalton: *Minimum periods, Modulo p, of First-Order Bell Exponential integers*; Mathematics of Computation, Vol.14, No.80, October 1962, pp. 416–423

[6] J. Shallit: *A Triangle for the Bell Numbers*; A collection of Manuscripts Related to the Fibonacci Sequence; 18th Anniversary Volume; The Fibonacci Association, Santa Clara, California, 1980, pp. 69-71

[7] N.D. Thomsson & R.P. Polivka: *APL2 in Depth*; Springer-Veriag, New York 1995

[8] ISO Document CD13571: *Programmming Language APL - Extended*; Committee Draft prepared by the APL Working Group ISO-IEC/JTC1/SC-22/WG3 Version 1; International Organization for Standardization ISO, Geneva, August 1993

[9] *Dyalog APL/W Reference Manual - Version 7*; Dyadic Systems Ltd. 1994

Appendix

As mentioned in the paper, Bell showed that for $n>0$ the exponential numbers may be calculated by the form:

$$B_n = \sum_{s=1}^{n} \frac{1}{!(s-1)} \left[\sum_{r=0}^{s-1} (-1)^r \binom{s-1}{r} (s-r)^{n-1} \right]$$

In fact, the form is the sum of the Stirling set numbers or Stirling numbers of the second kind $S(n,s)$:

$$B_n = \sum_{s=1}^{n} S(n,s)$$

with $S(n,s) = \frac{1}{!(s-1)} \left[\sum_{r=0}^{s-1} (-1)^r \binom{s-1}{r} (s-r)^{n-1} \right]$

A defined function *BELLS*, based on this algorithm, returning as explicit result the Bell number B_n, may be:

```
      ∇Z←BELLS N;R;S
[1]    →(+/N=0 1)/0,Z←S←1
[2]    END:R←¯1+⍳S←S+1
[3]    Z←Z+(-/(R!S-1)×(S-R)*N-1)÷!S-1
[4]    →(S<N)/END
      ∇

      BELLS 0
1
```

```
      BELLS 10
115975

      BELLS"0,⍳10
1 1 2 5 15 52 203 877 4140 21147 115975
```

The maximum of the domain of the right argument N is N = 137, with B_{137} = 1.36593938472514E172, as for N = 138 the alternate sum, before having been divided by !S-1, exceeds the largest number represenable 1.79769313486232E308. Benchmarks have been done for N = 40(40)129. Results are shown below (in ms):

N	40	80	120
BELLS N	34	127	311
BELLS"0,⍳N	551	3536	11967

As N increases, *BELLS* is from 17 to 113 times slower than *BELL4*. The algorithm is only of theoretical importance, but it is totally unsuited to the calculation of the Bell numbers.

Note: The algorithm treated in the Appendix may also be formulated as:

$$B_n = \sum_{s=1}^{n} \sum_{r=0}^{s-1} \frac{(-1)^r (s-r)^{n-1}}{(!r)(!s-r-1)}$$

It looks simpler, but a defined function based on this form is even slower than *BELLS*, by 24% to 33%.

Joseph De Kerf
Rooienberg 72
B-2570 Duffel
Belgium

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 normally 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: _____

Category (please tick box) to run from: 1st May ❑ August ❑ Nov ❑ Feb ❑

UK private membership	£12	❑
Overseas private membership	£14	❑
Airmail supplement (not needed for Europe)	£4	❑
UK Corporate membership	£100	❑
Corporate membership overseas	£135	❑
Sustaining membership	£430	❑
Non-voting UK member (student/OAP/unemployed only)	£6	❑

PAYMENT - in Sterling or by Visa/Mastercard/JCB only

Payment should be enclosed with membership applications in the form of a UK Sterling cheque to "The British APL Association", or you may quote your Mastercard, Visa or JCB number.

I authorise you to debit my Visa/Mastercard/JCB account

Number: ⌊_⌊_⌊_⌊_ ⌊_⌊_⌊_⌊_ ⌊_⌊_⌊_⌊_ ⌊_⌊_⌊_⌊_ Expiry date: ⌊_⌊_ | ⌊_⌊_

for the membership category indicated above,

❑ annually, at the prevailing rate, until further notice
❑ one year's subscription only

Signature: _____

Data Protection Act: The information supplied may be stored on computer and processed in accordance with the registration of the British Computer Society.

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.

2000/2001 Committee

Chairman	Adrian Smith 01439-788385 apl385@compuserve.com	Brook House Gilling East YORK YO62 4JJ
Secretary	Anthony Camacho 0117-973 0036 acam@tesco.net	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
Projects and Publicity	Dr Alan Mayer 01792-205678x4274 a.d.mayer@swansea.ac.uk	European Business Management School, Swansea University, Singleton Park, SWANSEA SA2 8PP
Webmaster	Ray Cannon 01252-874697 100430.740@compuserve.com	21 Woodbridge Road, Blackwater, Camberley, Surrey GU17 0BS
Activities	Jon Sandles 01904-612882 jon_sandles@csi.com	138 Burton Stone Lane, YORK YO30 6DF
Education	Dr Ian Clark 07931 370304 earthspot@aol.com	1, Heifer Mill Cottages, Mosterton, Beaminster, Dorset DT8 3HG.
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: Jonathan Barman (01488-648575), Anthony & Sylvia Camacho,
Ray Cannon (01252-874697), Marc Griffiths,
Bob Hoekstra (01483-771028), Jon Sandles (01904-612882)

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: 01653-696760
Fax: 01653-697719
Email: sales@causeway.co.uk
Web: www.causeway.co.uk

Compass Ltd
Compass House
60 Priestley Road
GUILDFORD, Surrey GU2 5YU
Tel: 01483-514500

Dyadic Systems Ltd
Riverside View, Basing Road,
Old Basing, BASINGSTOKE,
Hants. RG24 0AL
Tel: 01256-811125
Fax: 01256-811130
Email: sales@dyadic.com
Web: www.dyadic.com

HMW Trading Systems Ltd
Hamilton House,
1 Temple Avenue,
LONDON EC4Y 0HA
Tel: 0870-1010-469
Email: HMW@4xtra.com

Insight Systems ApS
Nordre Strandvej 119G
DK-3150 Hellebæk
Denmark
Tel: +45 70 26 13 26
Fax: +45 70 26 13 25
Email: info@insight.dk
Web: www.insight.dk

APL2000
Rapid Application Development
6610 Rockledge Drive
Suite 502
Bethesda MD 20817
USA
Email: sales@apl2000.com
Web: www.apl2000.com

Soliton Associates Ltd
Groot Blankenberg 53
1082 AC Amsterdam
Netherlands
Tel: +31 20 646 4475
Fax: +31 20 644 1206
Email: sales@soliton.com

Dutch APL Association
Postbus 1341
3430BH Nieuwegein
Netherlands
Tel: +31 347 342 337