

# VECTOR

## Windows and ISI APL

- APL\*PLUS II Version 5 reviewed 55
- Gfeller and Barman on APLIWIN 64
- Adams on Visual Basic 100
- Bowman on Internet 113

## ... plus

- APL93 Invitation & Abstracts 8
- 16-page Educational Supplement 27



*The Journal of the  
British APL Association*

A Specialist Group of the British Computer Society

ISSN 0955-1433

Vol.9 No.4 April 1993

## Contributions

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

Please supply as much material as possible in machine-readable form, ideally as a simple ASCII text file on an IBM PC compatible diskette (any format). APL code can be accepted as camera-ready copy, in workspaces from I-APL, APL\*PLUS/PC, APL\*PLUS II, IBM APL2/PC or Dyalog APL/W, or in documents from Windows Write (use the Vector TrueType font, available free from Vector Production).

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

## Membership Rates 1992-93

Category	Fee	Vectors	Passes
UK Private	£12	1	1
Overseas Private	£20	1	1
(Supplement for Airmail, not needed for Europe)	£8		
UK Corporate Membership	£100	10	5
Overseas Corporate	£155	10	
Sustaining	£430	50	5
Non-voting Member (Student, OAP, unemployed)	£6	1	1

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

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

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

## Advertising

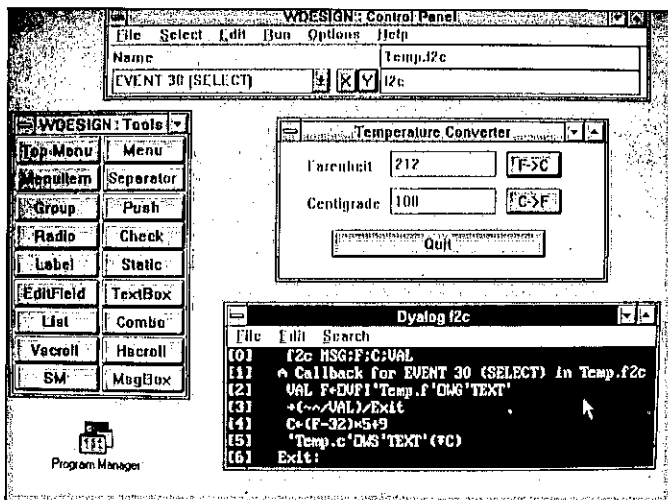
Advertisements in VECTOR should be submitted in typeset camera-ready format (A4 or A5) with a 20mm blank border after reduction. Illustrations should be photographs (b/w or colour prints) or line drawings. Rates are £250 per full page, £125 for half-page or less (there is a £75 surcharge per advertisement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

Deadlines for bookings and copy are given under the Quick-reference Diary. Advertisements should be booked with, and sent to: Gill Smith, Brook House, Gilling East, YORK YO6 4JJ. Tel: 04393-385.

# CONTENTS

		Page
<b>Editorial: What you know is what you love</b>	Jonathan Barman	3
<b>APL NEWS</b>		
Quick Reference Diary		5
APL Training Courses for 1993	Gill Smith	7
APL93 in Toronto		8
Conference Invitation		
Abstracts of Accepted Papers		
British APL Association News:		
Chairman's Report	David Eastwood	18
"The APL Review"	Alan Mayer	20
News from Sustaining Members	Gill Smith	22
<b>The Education Vector</b>	Alan Mayer	27
<b>REVIEWS SECTION</b>		
APL Product Guide	Gill Smith	43
APL*Plus II/386 Version 5.0	Dave Piper	55
A First Look at the ISI APL/Windows (APLIWIN) Beta Release	Martin Gfeller	64
ISI APL: Programming for Windows	Jonathan Barman	68
<b>RECENT MEETINGS:</b>		
APL-landing to Kronstadt island (an experience of APL operation)	from Pavel & Oleg Luksha	82
Roger Hui on "An Implementation of J"	notes by Anthony Camacho	85
<b>GENERAL ARTICLES</b>		
APL Experiences and MicroSoft's Visual Basic for Windows	Martyn Adams	100
Hooking up to the Internet	Dick Bowman bowman@apl.demon.co.uk	113
<b>TECHNICAL SECTION</b>		
The Challenge of the New	Duncan Pearson	120
Technical Correspondence		122
Sharing the Spoils or Circling the Square	Mike Day	123
An Exchange on Primes	Roger K.W. Hui	130
Span Representation:		
Improving the Display of J Verbs	Richard Oates	135
An APL Truetype Font	Adrian Smith	138
Index to Advertisers		143

# dyalog APL/W



## APL for Windows™

Dyalog APL/W is a true 32-bit Windows application and it shows! Click on the APL icon and up pops a powerful multi-window APL development system, with a host of features designed to enhance your productivity and enjoyment. But that's just the start. APL system functions provide you with simple and elegant, yet powerful tools to drive the Windows Graphical User Interface (GUI). In one statement you can create a GUI object such as a PushButton, specify its appearance and behaviour, and attach a "callback" function to be run when the button is pressed. There's even a WDESIGN workspace to help you. Windows programming has never been easier - and your applications will run unchanged under OS/2 and on Unix workstations in future releases of Dyalog APL.

## and it's friendly too

Dyalog APL/W supports Dynamic Data Exchange (DDE), using standard APL Shared Variables, so it can talk to any other Windows application that supports DDE. Can you imagine the power of APL combined seamlessly with your favourite databases, spreadsheets and other tools?

# DYADIC

All trademarks acknowledged.

# Editorial: What you know is what you love

*by Jonathan Barman*

The programming language that one learns really well seems to become etched in the mind, and it changes the way of thinking about programming for ever after. APL has unquestionably changed my way of thinking. Having changed one's way of thinking, however, one can get locked into that mode and become unable to appreciate other ways of thinking. This could be one of the main reasons for the difficulty in getting more people to program in APL.

APL programmers seem to be able to move to other languages much more easily than experts in other languages can move to APL. Changing from a scalar language to an array language is undoubtedly difficult. Unless there are clear advantages in programming in APL people are not going to be bothered to make the effort to change.

Over the past year I have had to learn three new languages and have developed substantial systems in each. The learning process is quite frustrating. Problems which are incredibly simple in APL seem to be extraordinarily difficult in the other languages. I am sure that the same remarks would be made by someone who is expert in another language, for example C, on attempting to program in APL. It is usually easier and faster to solve a problem in the language you know well and therefore love.

Converting programmers to APL is getting progressively more difficult. In the 70s and 80s the APL environment was so outstanding that it made an immediate impact on anyone used to the traditional compiled languages. This is no longer true. Most languages nowadays have excellent debugging facilities which match or better those available in APL.

The effort to disseminate APL to students seems very important. Having got someone to use and know APL it should enable them to program better in any programming language. The British APL Association is making the effort with Education Vector, which has a wider circulation than Vector. I-APL and J are freely available. If we don't manage to promote APL in this way, I fear that APL will gradually die and become one of the many lost and forgotten languages.

I think that the APL community needs to be aware of what is going on in the outside world, and what the competition looks like. In this issue there is an article by Martyn Adams about Visual Basic. Martyn is an expert APL

programmer and has moved to Visual Basic because of market pressure, so his views are of interest. What comes across most clearly is the way in which Visual Basic concentrates on allowing people to implement simple Windows systems without having to do any serious programming.

Most APLers love programming, but the majority of people hate it. Visual Basic, in company with 4GL languages, is aimed at the mass market with the superficially attractive premise that programming is not really necessary, but when it is necessary the programs are easy to write and understand. This is, of course, hogwash. Any medium-sized system solving a real problem has to have tens of thousands of lines of the 'easy' code, and the sheer volume then makes it difficult to understand.

Visual Basic is becoming quite popular. The environment appears to be better than many APL interpreters. Even if you convince someone that APL is a much better language than Basic, the ability to create quick simple Windows programs is liable to be a significant factor in their decision to favour Visual Basic.

Fortunately, all is not gloom. David Piper reports that Manugistics have produced a usable Windows interface with Version 5 of APL\*PLUS II. Dyalog APL has an excellent Windows interface. MicroAPL have had extensive experience for many years with their APL on the Macintosh. With these new tools APL should become one of the top contenders again.

## ***Vector Back Numbers***

Back numbers of Vector are available from:

**British APL Association  
c/o Gill Smith  
Brook House, Gilling East,  
YORK YO6 4JJ**

Price in UK: £12 per complete volume (4 issues);  
£14 (overseas), £20 (airmail) including Postage.

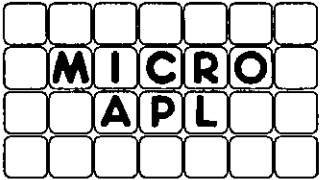
## Quick Reference Diary 1993

Date	Venue	Event
23 April	IEE London	Neural Networks for Modelling and Forecasting: Michael Bramson,  Forecasting and Planning: Maurice Jordan
11 June	IEE London	<b>AGM</b> Presentation of Outstanding Achievement Award  Educational Software for the Macintosh: Ian Clark APL in Education: Richard Weber.
August 15-19 1993	Toronto, Canada	APL93
17 Sept	IEE, London	Vendor Forum
26 November	TBA	APL Professional Development Workshop: This will be an all-day event (lunch is included) - a small fee will be payable

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

### Dates for Future Issues of VECTOR

	Vol.10 No.1	Vol.10 No.2	Vol.10 No.3
Copy date	4th June 93	3rd Sept 93	3rd Dec 93
Ad booking	11th June 93	10th Sept 93	10th Dec 93
Ad Copy	18th June 93	17th Sept 93	17th Dec 93
Distribution	July 93	October 93	January 94

—	APL Preferences	▽										
<h1>APL.68000</h1>												
												
<table border="1" style="width: 100%;"><tr><td style="text-align: center;"><b>Platforms</b></td><td style="text-align: center;"><b>Versions</b></td></tr><tr><td><input checked="" type="checkbox"/> RISC System/6000</td><td><input type="radio"/> Level I</td></tr><tr><td><input checked="" type="checkbox"/> Apple Macintosh</td><td><input checked="" type="radio"/> Level II</td></tr><tr><td><input checked="" type="checkbox"/> Commodore Amiga</td><td></td></tr><tr><td><input checked="" type="checkbox"/> Atari ST</td><td></td></tr></table>			<b>Platforms</b>	<b>Versions</b>	<input checked="" type="checkbox"/> RISC System/6000	<input type="radio"/> Level I	<input checked="" type="checkbox"/> Apple Macintosh	<input checked="" type="radio"/> Level II	<input checked="" type="checkbox"/> Commodore Amiga		<input checked="" type="checkbox"/> Atari ST	
<b>Platforms</b>	<b>Versions</b>											
<input checked="" type="checkbox"/> RISC System/6000	<input type="radio"/> Level I											
<input checked="" type="checkbox"/> Apple Macintosh	<input checked="" type="radio"/> Level II											
<input checked="" type="checkbox"/> Commodore Amiga												
<input checked="" type="checkbox"/> Atari ST												
<h1>APL for GUIs</h1>												

**MicroAPL Ltd.**, South Bank Technopark, 90 London Road,  
LONDON SE1 6LN, UK

Tel: 071 922 8866 Fax: 071 928 1006

Applelink: microapl Internet: microapl@applelink.apple.com



## APL Training Courses for 1993

For confirmation of dates and/or further details please contact the vendor directly.

Level	Company	Days	Dates
Beginners	MicroAPL	1	8 April, 27 May, 22 July, 16 Sept, 11 Nov
APL*PLUS/PC	MicroAPL	1	6 May, 24 June, 19 Aug, 14 Oct, 9 Dec
Introduction to APL2	MicroAPL	1	20 May, 15 July, 9 Sept, 28 Oct
APL*PLUS II/PC Conversion	MicroAPL	1	13 May, 8 July, 26 Aug, 21 Oct, 16 Dec
Intermediate	MicroAPL	1	15 April, 10 June, 5 Aug, 23 Sept, 18 Nov
Advanced	MicroAPL	1	22 April, 17 June, 12 Aug, 7 Oct, 25 Nov
Statgraphics	Mercia	1	20 April

If you would like to have your courses or seminars listed in Vector, please contact Gill Smith with the details.

# The International Conference on APL August 15-19, 1993

University of Toronto, Ontario, Canada

## Invitation and Preliminary Program

### Welcome: Conference Chair

In the Spring of 1973, a conference called APL5 was held in Toronto at the Inn on the Park. The banquet was held at the Ontario Science Centre, and the guest speaker was Dr. Kenneth Iverson, the inventor of APL. Twenty years later, Toronto will again host an international APL conference. This time it will be held in the Medical Sciences Building of the University of Toronto, which is near the centre of the city.

I am delighted that Professor Leroy J. Dickey has volunteered to be the Program Chairman for the conference; he has provided a rich and varied formal program. Further, Bob Bernecky has volunteered to take responsibility for all the other program events which do not involve the formal delivery of papers. These include tutorials, workshops, panels, poster sessions, birds-of-a-feather sessions and an APL Art Gallery, all to round out and enrich the program. There will also be an exhibitor's area which we expect will be well supported by vendors.

Toronto in August is a delightful place to be. It is a safe, walk-about city, with literally hundreds of activities within a short walk of the conference site, including sightseeing, shopping, dining and theater. Delegates are encouraged to bring their families, and a variety of activities will be available for accompanying persons. Toronto is easy to reach, having an international airport with direct flights from many European, American and Canadian cities. It is also within a day's drive of Boston, New York, Washington, Pittsburgh, Cincinnati and Chicago.

I look forward to greeting you at the APL93 Conference in Toronto.

*Larry Moore*  
*Conference Chair*

## Program Chair

Hello. I am pleased to report to you that the papers for this year's conference are among the best that I have seen for any APL meeting in the last ten years. I find the range of topics particularly exciting, as I am sure you will too, when you browse through the attached list. I am impressed at the wide variety of applications to which APL has been put both in research and in production, in science, industry, business and academia. Not only are the applications of APL being developed, but the foundations of APL are being explored as well. There are new areas of research, and new linguistic features. Are these things just a flash in the pan, or are they elements that will join the mainstream of computing and be taken for granted in years to come? We leave that question for time to resolve, but we invite you to be a part of the fomentation and excitement now.

Come, attend the invited talks, attend the contributed papers, participate in a tutorial, participate in a workshop, bring a poster, bring some shareware, participate in a Birds of a Feather session, and have a good time. A friend of mine says that the best part of a conference is the people that one meets in the halls, outside of sessions. For me too, a high point of APL meetings, is the people that I meet.

Come to APL93, and let us meet and share ideas about APL.

*Leroy J Dickey*  
*Program Chair*

## Tutorials Chair

Welcome to Toronto! APL93 represents a dramatic improvement over previous conferences — we have encouraged a varied and plentiful selection of tutorials, fully integrated with the remainder of the technical program, to enhance your educational opportunities while in Toronto. We have rethought a number of other conference activities, including posters sessions, the APL art gallery, the APL software exchange, Birds-of-a-Feather sessions, with an eye toward enriching your experience here, by changing their focus to be of greater benefit to attendees and to perform outreach to the community beyond that of the APL community.

Do you feel that tutorials at APL conferences are somehow peripheral to the main program? Have you ever been frustrated by short presentations of research papers, in which time constraints prevent The Really Interesting Questions from being answered. Do you want to have the time to actually learn how some of these new gadgets work?

APL93 has turned that whole idea on its head, by making tutorials a well-integrated part of the conference, open to all attendees. Yes, there is the ever-present hazard of coming away from APL93 knowing more than you did when you arrived, but I know you'll get accustomed to it and enjoy it!

The tutorial program for APL93 offers a wider selection of topics than any previous APL conference, and registration for two half-day tutorials is included with your pre-registration fee. We are making attendance at tutorials possible at very low cost, because we want to make education an integral part of the conference program.

*Robert Bernecky*  
*Tutorials Chair*

## Tutorials

Many tutorials this year will be hands-on, with each attendee sitting at her or his own PC or SUN workstation, provided by the University of Toronto Department of Computer Science. Subjects (and speakers) will be:

- Teaching Calculus — Kenneth E. Iverson
- An Introduction to J — Donald McIntyre
- Image Processing in APL — Manuela Schäfer, Dinu Scheppelmann
- APL for Actuaries — Richard L. Vaughan
- 3D-Visualization of Medical Images in APL — Pitt Meinzer
- AGSS: A Graphical Statistical System — Peter A.W. Lewis
- An Introduction to Parallel Computing — Robert Bernecky
- Mathematica for APLers — Richard J. Gaylord
- An Introduction to APL — Ben Best
- Windows GUI Programming in APL — Eric Iverson
- Wicked Problems and APL — Chris Lee

Please be advised that registration at specific tutorials will be opened to non-registrants on a space-available basis on the day of the tutorial. I believe that such outreach is an important activity, particularly when APL has a vital role to play in the emerging area of massively parallel computing.

Therefore, to avoid disappointment because of Standing Room Only crowds, please be sure to register for your tutorials at the time of pre-registration. Otherwise, you may be left out, due to lack of space. In particular, hands-on

tutorial space is limited by machine availability. Pre-registration will let us find appropriate venues for the tutorials so that you won't be left out or sitting on the stairs.

The registration form allows you to specify alternate choices for each tutorial timeslot, in case the tutorial you prefer is fully booked. Also, you'll save money, because early pre-registration offers you two free tutorials, whereas you only get one if you register late.

Most of the tutorials will be offered on Sunday, August 15th and again during the conference, for the convenience of conference delegates. I am very grateful to those presenters who have volunteered to take on this extra workload!

*Robert Bernecky*  
*Chair of Tutorials*

## Invited Speakers

### Opening Plenary:

#### **The Triumph of Symbols over Words**

*Professor Donald McIntyre*

Professor Donald McIntyre illustrates highlights in 5,000 years of evolving notation. Ideas, once expressed in rhetorical form, are increasingly represented by symbols. Early forms of symbolism were often subsequently modified or replaced. The new symbolism of Iverson and Hui's J language provides a reason for reflecting on this history

Donald McIntyre is a retired Professor of Geology who has been using computers and teaching computer programming for over 30 years. He has been an invited speaker at many APL Conferences, and was a Distinguished Lecturer for ACM from 1985-1990. He has given workshops on J in London and in many cities in the United States and Canada. He contributed one of two papers on J included in the IBM System Journal's special issue marking the 25th Anniversary of APL. His 70th birthday will coincide with his Sunday presentation of this tutorial.

## Closing Plenary:

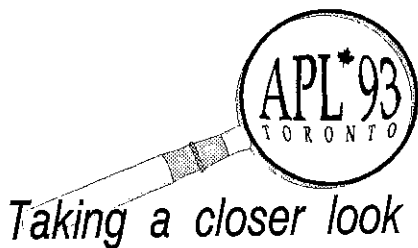
### Modelling Petroleum Chemistry in the Era of Clean Fuels

*Dr. Stephen B. Jaffe*

Today's environmental issues require the Oil Industry to make a product of specific chemical composition and specific physical properties. To meet these new targets, petroleum refiners and planners must turn to mathematical models for guidance. The challenge for these models is to follow the chemistry of thousands of molecular species through hundreds of thousands of chemical reactions. A new approach called Structure Oriented Lumping (SOL) has been developed which represents petroleum molecules as collections of structural increments. Models using SOL are capable of predicting far more detail than ever before.

APL plays a crucial role in every aspect of Structure Oriented Lumping. Experimental data is organized, reduced, and analyzed with APL. Reaction networks are generated using a rule-based approach for specifying the appropriate chemistry with APL. Finally, FORTRAN code is automatically generated by APL to integrate the pertinent differential equations. This novel solution of a major problem is a direct consequence of our long-time familiarity and use of APL.

Stephen B. Jaffe has been an active APLer for more than twenty years. He has authored many papers on APL, served on the ACM SIGAPL Board, has organized APL conferences, and taught APL. Dr. Jaffe is a Senior Research Consultant at Mobil Research and Development in Paulsboro, New Jersey.



## Abstracts of Accepted Papers

### *APL Helps Deaf Person to Hear Again* —

P. Destauriers. This paper demonstrates how APL can be an effective tool in the world of research and development of electronic systems by describing the use of APL in the research and development of a cochlear implant device. The device described is a biomedical electronic system which permits a completely deaf person to regain partial hearing capabilities. The paper begins with an overview of the system itself, followed by a discussion of how APL was utilized.

*APL Programming without Tears* — P. Naeve, B. Strohmeier, and P. Wolf. The authors discuss some of the drawbacks involved in producing APL code, including the lack of general understanding about APL primitives, the lack of familiarity with APL glyphs, and the awkwardness of the traditional process for writing APL code. This last aspect is discussed in detail, with the proposal that Knuth's idea of iterate programming be adapted to APL to produce more readable code. An example is provided to highlight the merits of this application.

*Approaching Classical Algorithms in APL2* — F. Grimm and Manspeter Bieri. Classical algorithms and data structures as made popular by Knuth's *The Art of Computer Programming* have been largely ignored so far by APL2 programmers. This was understandable to a certain degree in the era of APL, but has lost its justification with the rise of APL2 and similar APL successors. First, the essence of classical algorithms as well as the principal ways of data structuring and accessing are re-examined. Then it is shown how pseudocodes or implementations of classical algorithms can be transferred one-to-one into APL2 almost automatically and often with an acceptable resulting performance. But, in general, the better way consists in maintaining only the general design principle behind a classical algorithm and combining it with the elegant and efficient specific possibilities of APL2. Both approaches are explained and illustrated by means of a typical classical algorithm from picture processing.

*APROCL: A Hybrid Language* — Dennis Holmes and John E. Howland. This paper describes the design of a hybrid language that combines the features of an array processing language and Lisp dialect in a consistent and useful manner. This language, APROCL (Array PROcessing Lisp) is derived from the J dialect of APL and the Scheme dialect of Lisp. The base syntactic structure is taken from Scheme, while the array processing features are based on the J programming language. A prototype implementation has been made and some experiences with this implementation are described. This implementation uses J as an embedded array processing engine in a Scheme interpreter / compiler.

The language as specified provides a set of data types and manipulation tools that is more diverse than found in either Scheme or J. APROCL allows the programmer to apply array processing functions to lists of arrays in the Scheme style and list processing functions to arrays of lists in typical J style. The result is a language that not only brings array processing capabilities to Scheme, but also significantly extends the functionality of the Scheme language.

*ARDA: Expert System for Reliability Data Analysis* — Jake Ansell and Julhim Al-Docri. This paper explores the design and implementation of ARDA, an Expert System to analyze reliability data. Initially the knowledge domain is explored for the viability of the system. The philosophy of design of the system is discussed. Details of the implementation are described. There is discussion of extension of the system to other statistical analyses and of using alternative inferential bases.

*Array Morphology* — Robert Bernecky. Array morphology is the study of the form, structure, and evolution of arrays. An array annotation for a program written in an applicative array language is an abstract syntax tree for the program, amended with information about the arrays created by that program. Array notations are useful in the production of efficient compiled code for applicative array programs. Array morphology is shown to be an effective compiler writer's tool. Examples of an array annotator in action are presented, showing its value in array morphology. Array morphology is shown to provide methods for static detection of certain classes of programming errors.

Assertions, a generalization of declarations, offer significant benefits to application writers as well as compiler writers. Assertions, and therefore declarations, may be represented as conjunctions, and are, therefore, conforming extensions to ISO Standard APL. A domain conjunction is offered as an example of how assertions might be defined.

Several other points are made relevant to the implementation of array language compilers and interpreters.

*Bayesian Methods in APL* — Alan Sykes, A. Mayer, and T. Stroud. This paper explores the ideas of Bayesian Statistical Inference from the point of view of statistical computation. As such the paper is largely self-contained, enabling those more familiar with APL than statistics to understand the basic statistical methodology and appreciate its strengths. The paper further reports on recent collaborative work in providing a computational framework in APL for extending Generalized Linear Models to their Bayesian counterparts. This work relies heavily on methods for numeric integration combined with the iteratively re-

weighted least squares procedure at the heart of the Generalized Linear Models as incorporated into ASLREG, the first volume produced in ASL (the APL Statistics Library).

*Building the APL Atlas of Natural Shapes* — Gérard A. Langlet. It was previously shown that APL contained the most powerful idiom  $\neq \backslash$  that could be used, directly in the language all computers know: binary algebra, to build models as well for physics as for biology and computer science. Several papers on the subject were published or submitted inside the APL world as well as outside. The purpose of this paper is to show how a classical model, built to generate fractal shapes in plane geometry (2-D) can be revisited and considerably extended, thanks to the properties of  $\neq \backslash$  and of array-oriented binary algebra.

*Compiling APL to SISAL* — Robert Bernecky. Compiling APL is problematic. The freedom APL grants to programmers, to dynamically redefine the type, rank, and shape of an array, and to redefine functions on the fly, is antithetical to the strict requirements of traditional compiler design. That freedom also happens, perhaps not coincidentally, to make iterative APL code run hundreds of times slower than compiled traditional code. SISAL, an applicative array language, has solved several of the problems of compiling array languages, and is beating Fortran on a number of supercomputer benchmarks. This paper presents the challenges of compiling APL programs into SISAL and recommendations for solving some of the problems which occur in such a translation. Several benchmarks are proposed for comparing interpreted and compiled APL.

*Confessions of Two APL Educators Learning J* — Murray Eisenberg and Howard Peelle. The paper reports how two university teachers of APL began to learn J. By presenting accounts of a series of small experiments, it reveals our understandings and misunderstandings along the way. It discusses things we especially liked and disliked about J and the resources available for learning it, and indicates some possible implications of our experience for teaching J.

*Co-operative Programming with Windows DDE* — Adrian Smith. In the past, there have been two kinds of APL applications: one-off investigative developments, and major vertically integrated applications. The first kind are usually characterised by a small volume of powerful APL code, precisely targeted towards exploring an interesting business idea. The second kind are typically wrapped around with a great blanket of user code, which is all about the entry, validation, and presentation of data.

With the advent of MS Windows (and the inevitable move of the PC community toward Windows applications) we can begin to exploit the inherent power of Word and Excel as native components of our APL development. This paper begins the exploration, with just two simple ideas: an APL button-bar for Excel (for Excel freaks who have begun to want a real array

language), and an Excel-based formatting tool (for APL freaks who get bored with laying out neat reports).

*Discriminant Analysis* — Eero Korpelainen. Abstract not available at time of printing.

*Distributed Computing in the Workstation Environment* — Johann Mithöner. Exploiting unused processor time in workstation environments can result in large performance gains in user applications. Techniques for implementing client-server based distributed processing are shown. A time-consuming algorithm is sped up by using idle workstations in a local area network as APL servers.

*Efficient Maximum Likelihood Estimation of Linear Models with APL* — Frank C. Ripley. Maximum likelihood estimators used in statistics and econometrics have desirable properties; however, due to the complexity of their solution maximum likelihood techniques are not widely used. This paper examines an APL implementation of one of the most efficient algorithms used to estimate the parameters of the univariate ARMA process. The straightforward use of APL is found to be unacceptable and so a systematic search for optimization is made. This search results in an approximate solution for the ML estimator, the use of *DATA* and FORTRAN, and special matrix techniques to increase the efficiency of the algorithm. The matrix techniques used are implementations of sparse, banded, and block diagonal data structures. Additional matrix techniques involve incremental updating of matrices. The effect of these optimizations is to bring the computational cost from an  $O(N^3)$  problem to an  $O(N^1)$  problem. This translates into a significant reduction in computer requirements. For example, to estimate a six-parameter ARMA model with 5000 observations, memory requirements fall from approximately 95 Mb to 300 kb. Computer time falls from about 400 years to about 2 minutes.

*Extending APL2 to Include Program Control Structures* — David A. Selby. APL language designers have always argued that implicit control flow beyond that of the branch arrow and array operations is not required in the APL2 language definition. On the other hand, computer scientists object to APL on account of the lack of control flow within its basic language definition. The first thing a reader looks for in trying to understand a program in any language is its control flow. Given the wide variation in conditional branch idioms in APL2, and the lack of line blocking, it is often difficult for even a skilled reader to get a clear understanding of a previously unseen APL function.

This paper proposes the addition of various well-established program control flow structures into the APL2 language as reserved words.

*Extending the APL Character Set* — James A. Brown, Brent Hawks, Ray Trimble. APL is often presented as a notation that is independent of national language because of its symbolic nature. Paradoxically its unique character set has led to APL being treated as if it were itself a national language. This has meant, in many



practical situations, that the APL character set is incompatible with national language character sets.

IBM's APL2 attempted to avoid these problems by defining a set of extended (31 bit) characters, and this has indeed been helpful in handling Asian languages such as Kanji. But the character mappings adopted have not been adequate to support all European and Middle Eastern language characters, nor have they been consistent across platforms.

With the advent of international standards for character assignment, there is an opportunity to create APL systems that handle the character sets of the world in an efficient and elegant manner. While at first the solution appears to require only a recitation of code point assignment, an analysis of the problem leads to interesting and disturbing questions about a number of APL system functions, system variables, commands, and file facilities. This paper explores problems with internal conversions, external representation, migration, compatibility, and interplatform portability.

*Extending the Two-Partner Shared Variable Protocol to n Partners* — Thomas Kolarik. APL as a language has always provided a high level of abstraction to its users. Only in the field of communication APL channels — called shared variables — have been restricted to at most two partners. As most distributed applications demand more than two interacting programs, APL users had to use their own techniques, like client-server, to develop applications that share information among more than two partners. This paper presents a proposal to move the administration of shared memory from application to language level: an extension to the well-known shared variable protocol to share memory among more than two partners. A programming technique — replicated workers — and a sample application for students that learn APL in a working group are shown that take advantage of this feature. These are both based on a prototype implementation of the proposed protocol.

*From Trees into Boxes* — David H. Steinbrook and Eugene McDonnell. This paper is a progress report on work undertaken to include tree data structures by means of the boxed data type available in J. Methods for displaying these boxed arrays as trees are shown. This work is part of a larger effort to provide a comprehensive set of facilities in J for working with tree structures. The facilities described were at first modelled in J and subsequently translated into C, in order to provide a J interpreter which has trees as native facilities. Thus this work also exemplifies the way in which one can tailor the J interpreter to special needs.

*Identification of Parallelism in Neural Networks by Simulations in the Language J* — Alexei N. Skurikhin and Alvin J. Surkan. Neural networks with back-propagation training are designed and expressed in the language J, an APL derivative with very powerful function encapsulation features. Both the languages J and APL help to identify and isolate the parallelism that is inherent in network training algorithms. Non-critical

details of data input and derived output processes are de-emphasized by relegating those functions to callable stand-alone modules. Such input and output modules can be isolated and customized individually for managing communication with arbitrary, external storage systems. The central objective of this research is the design and precise description of a neural network training kernel. Such kernel designs are valuable for producing efficient reusable computer codes and facilitating the transfer of neural network technology from developers to users.

*Introduction to Log-Linear Analysis and Implementing Newton-Raphson Algorithms in APL2* — Duncan McArthur. This paper introduces the method of log-linear analysis for performing hypothesis testing on contingency tables. The reader will see a brief development of this topic beginning with Pearson's classic chi-square test and using examples of analyses on two- and three-dimensional tables. The idea of hierarchical models and backward elimination are discussed. Finally, the APL2 implementation of the Newton-Raphson algorithm is described. Newton-Raphson is an iterative procedure for finding the roots of a function. It is used in log-linear analysis to find the maximum likelihood estimation of expected frequencies which cannot be calculated from expressions in closed form.

*JVOX* — David G. Smith and Joey K. Tuttle. It is easy to learn J; just talk to your computer. Voice recognition technology is already on the desktop. You can use voice input to bypass some common obstacles and promote gratifying and enjoyable experimentation with the language. It is possible to avoid pawing through references to find an elusive construct. Just talk, and anything from simple primitives to complex idioms can appear on your computer screen. Spelling and typing errors are all but eliminated, and talking constantly reinforces the new notation's meaning.

This paper presents results from some experiments with a JVOX prototype. The presentation includes a demonstration.

*Learning Modern Algebra* — Pavel Luksha. The study of modern algebra, especially that of finite fields and their application to error codes is aided by the use of a computer. This paper is an account of how a high school student put APL to good use in a self-study course in this University level subject. There is a review of finite algebra, a short discussion of finite fields, a discussion of polynomials over those fields, and their use to construct error-detecting and error-correcting codes.

*Modern APL Windows* — Richard R.N. Eller. Windows 3.1 represents the biggest revolution to APL technology since the advent of full-screen techniques. This creates a new challenge to APLers migrating their applications from mainframe or PC-DOS environments into Windows 3.1. This paper describes an easy means to adapt existing and new applications to exploit Windows Graphical User Interfaces (GUI). By using the techniques described below one can utilize most GUI

features without needing to comprehend the massive amount of detail typical of Windows programming. Additionally, any application will also be backwards compatible to the DOS character-based screen interface.

*A Parallel Topological Feature Map in APL* — J. Frey, D. Scheppelmann, G. Glombitza, H.P. Meinzer. One can distinguish two different approaches to neural networks. The supervised networks and the self-organizing or unsupervised neural networks. The first type of neural nets is supplied with an ideal result regarding the input. During the learning procedure the neural net adjusts weighting factors of the links between neurons so that the input feature vectors map to the ideal output. Those nets are used for example in robotics, where the ideal result is well known: it is the position the robot should be placed in. For the cases where no ideal result is known, the second type of neural nets, the so-called self-learning Topological Feature Map (TFM) is appropriate. This paper will introduce such a neural net based on the idea of Kohonen's TFM. The original algorithm was extremely sequential and therefore not suitable for an APL implementation. The parallelization of the algorithm led to important improvements regarding speed and convergence to the global optimum.

*Point-wise Calculus* — Walter G. Spunde. The work of Richard Neideringer implementing differentiation in APL as a vector arithmetic is reformulated and extended, for functions of a single variable, to nested vectors whose components hold the values, at any number of given sample points, of a function and its derivatives up to any specified order. It is argued that, for teaching purposes, this sampling provides a more intuitive introduction to mathematical functions and the rules of calculus than do algebraic formulae, and that for certain calculations (such as the computation of polynomial approximations of high degree) the formulation provides superior algorithms for computation. As such, it offers an alternative approach to the teaching of elementary college mathematics.

*Roles of APL in Military Satellite Surveillance* — Jack G. Rudd. APL has had a significant prototyping role in military satellite surveillance for more than two decades. This paper begins by describing the success of the Defense Support Program in alerting the Patriot antimissile batteries of incoming Scud missiles in the Persian Gulf War. Subsequently it describes the prototyping role of APL on the Defense Support Program and other satellite surveillance programs. Several observations are made along the way contrasting project development methods involving APL prototyping with other methods of project development. The paper concludes by describing two new efforts: one which features rapid development and delivery of APL2 analysis programs on workstations for direct use by military customers; and one which uses APL2 to study and prototype specific military applications for potential implementation on a massively parallel processor.

*Rolling Dice: Some Notes on J and Teaching Probability* — Keith Smillie. This paper shows how J might be used in a course in elementary probability. All

examples are concerned with rolling dice, but the dice may have  $n$  sides, and be used to play music!

*SCARFS: An Efficient Polynomial Zero-Finder System in APL* — Tien Chi Chen. This paper introduces SCARFS (Symmetric Cluster-Adapted Root-Finding System), a new efficient APL workspace for general polynomial iterative zero-finding, as a practical outcome of our study of global iterative zero-finding. The 25 functions in this workspace are coordinated by the SCARF function with iterations performed by the function SCATTER (Symmetric Cluster Adapted ITERation).

*Solving Polynomials in Two or Three Variables* — R.G. Selfridge. An algorithm is described that allows two polynomials in two variables to be reduced to a single polynomial in one variable, and then back-solved to get all sets of solutions. The algorithm works faster than most other algorithms within its range of utility (providing sets of solutions as large as 70), and can be extended to cover some sets of three polynomials in three variables (those where one of the polynomials has only two of the variables). While subject to the known stability problems of polynomial root-finding, this algorithm can also be extended to provide for reducing to the single variable polynomial with symbols, thus permitting proof of results, and potential removal of extraneous roots. Such manipulation can then materially shorten the numerical process if the problem is to be applied to a number of different cases.

*Structured APL: A Proposal for Block Structured Control Flow in APL* — Robert Wilhoft. APL, although a very powerful language, has failed to gain wide acceptance in part due to its lack of control structures. A proposal is made for introducing structure in APL objects by adding to the items that can be located on the left side of the colon (:). These markers show the beginning and end blocks of code and allow for selection, iteration, and termination. The set of control structures is shown to be robust by showing their use in the creation of traditional control structures. A method of conversion to ISO APL is also presented.

This paper briefly discusses proposals that have been made in the past along with an evaluation of their merits and shortcomings. The paper ends with a discussion of related issues, including elimination of branching, introduction of definition blocks, and lexical scoping in APL. Several areas for continuing work are given.

*Structuring Functions with Operators* — David Eastwood. This paper draws its inspiration from a theme that has been popular at APL conferences over the years. This is an attempt to introduce some or all of the elements of structured programming into the APL language either via extensions to the language or via the adoption of some set of programming standards. In view of the speed and frequency with which agreed extensions to the APL language take place, it must be a reasonable assumption that there is no immediate likelihood of APL acquiring new flow control primitives. The programming standards approach relies heavily, however, on the willingness of the programmer to follow

them and much good work can be undone by a few minutes of hasty coding to fix a last-minute error.

*Talking with APL via DDE: Teaching an Old Dog New Tricks* — Steven J. Halasz and Andrei Kondrashev. Shared variables and auxiliary processors are well-known techniques for connecting alien applications and facilities to APL. The use of APL in the PC windowing multitasking environment of MS Windows, which incorporates a messaging model for interprocess communications, requires some new approaches to the implementation of shared variables and auxiliary processor support. Any such implementation should realize the maximum opportunities that the windowing environment offers.

This paper presents a shared variable interface to MS Windows graphics written in C and a simple charting package called TinyPlot written in Dyalog APL for Windows. The purpose of this paper is to demonstrate and investigate the Dyalog APL shared variables and auxiliary processor facilities, and to offer observations and conclusions about the benefits, limitations, and potential of DDE and DLL interfaces generally.

*The Testing of an APL Compiler* — Wai-Mee Ching and Alex Katz. The testing of the APL-to-C compiler, COMPC, developed at T.J. Watson Research Center consists of two components: a testing suite of 146 APL programs collected from various sources covering a variety of fields, and a unit-testing procedure which tests each primitive function on all possible subcases arising from different combinations of storage types and shapes. The second component, unit-testing, is an interesting example of the productivity APL can provide for software development. The unit-testing procedure is based on a workspace written in APL and utilizes the `⌈FX` feature of APL to create a test function dynamically from one of several templates. The testing of both components is automated through the use of control programs written in a command language under the VM/CMS operating environment.

*Transfinite Nesting in Array-Theoretic Figures, Changes, Rigs, and Arms. Part I* — Trenchard More.  
*Transfinite Nesting in Array-Theoretic Figures, Changes, Rigs, and Arms. Part II* — Trenchard More.  
Abstract not available at time of printing.

*Adaptive Learning Networks in APL2* — Alexander O. Skomorokhov. Adaptive Learning Networks (ALN) method is a tool to solve the problems of modelling, prediction, diagnosis and pattern recognition in complex systems. ALN is similar to neural network techniques, with the main difference being self-organization of network structure on the base of generation and estimation of various nodes, connections, and weights. A set of functions presented in the paper shows that ALN are simply realized in APL2. User-defined operators are used as a very convenient tool for ALN programming. The paper discusses application of implemented software to the problem of Burnout Heat Flux prediction in nuclear reactors. It is shown that the

ALN technique allows prediction of Burnout Heat Flux with approximately three times better accuracy than commonly used methods.

*Understanding ANOVA the APL Way* — Norman Thomson. Statistics textbooks expound the collection of techniques known as **analysis of variance** by using mathematical formulae. The term analysis of variance is something of a misnomer, since any analysis follows a primary exercise of partitioning sums of squares, that is entirely a matter of computation and data reorganization of the sort at which APL2 is uniquely adept. The crux of the analysis stage then consists of making choices between what is often a bewilderingly large range of subtly different sums of squares partitions. The primitive function `encl` with axis maps naturally into the data manipulations involved in sums of squares partitions for arrays. To the casual reader the latter are obscured rather than clarified by the use of mathematical formulae alone. This paper explores the relationship between APL2 functions and ANOVA, and goes on to illustrate how these can be used in the context of a designed experiment.

*Using Defined Operators and Function Arrays to Solve Non-linear Equations in APL2* — Stephen M. Mansour. There are many mathematical algorithms such as Newton's method used to calculate solutions to non-linear equations. This paper will show how easy it is to implement these algorithms in APL2 with a minimum of code using the defined operator and careful design of its input functions. The emphasis in this paper is on the method of developing an application using APL2 and the concepts of defined operators and function arrays. This requires that input functions be robustly designed to take arrays as arguments and to produce array results much like primitive scalar functions. In this manner one can minimize the number of calls to the input function and take advantage of APL2's array handling capabilities. Since operators are permitted at most two operands, it is critical to minimize the number of input functions. This requires the input function designer to think in terms of function arrays. Although function arrays are not yet part of the language, they can easily be defined in most any version of APL.

*The Workspace Manager: a Change Control System for APL* — Rexford H. Swain and Daniel F. Jonusz. This paper describes the Workspace Manager (WSM), a tool that helps to support and add discipline to APL system development and maintenance efforts. The WSM acts as a repository of APL objects (variables, functions, and operators). Programmers use WSM tools to find where objects are used, edit objects, save changed objects, and request that objects be installed into (or erased from) production workspaces. Periodically, the WSM installs new releases of production workspaces by merging new and changed objects into existing workspaces. Audit trails are maintained for all of these activities, making it possible to review the change history of an object, compare different versions of an object, compare different releases of a workspace, revert to an old release of a workspace, and so on.

## British APL Association News

### BAA Chairman's report - Activities

It is with some regret that I must report Peter Donnelly's resignation as Activities Officer of the BAA. Peter has served on the committee of the BAA for a number of years, notably as my predecessor as Chairman. As the current Chairman I can fully appreciate the time and effort Peter devoted to the BAA Committee and I would like to record our thanks for his many valuable contributions. Peter unfortunately feels unable to continue as Activities Officer due to pressure of work and has stepped down from the position with effect from February 1993. Pending the election of a new activities officer, I will be holding the fort myself.

In addition to finding a new Activities Officer, I am keen to enlarge the activities working group which supports our Activities Officer. I shall take this opportunity to describe the work that lies behind the activities programme of the BAA in order to stimulate the flow of willing volunteers.

Experience has shown that in many respects the most difficult part of the Activities Officer's job is to come up with ideas for meetings — both for topics and speakers. The danger of having one person do all the work is that we are thereby limited in the range of topics covered. The prime role of the activities working group is to generate helpful suggestions about meetings and, if appropriate, to approach potential speakers. The current activities working group (comprising myself, Misha Jovanovic, Duncan Pearson and Anthony Camacho) have each arranged one meeting this year by agreeing a theme and finding the two or three speakers required.

At the time of writing, we have arranged the remainder of the 1993 programme and are now considering the 1994 programme. You should note that the committee of the BAA has no hard and fast views about the preferred format for meetings and we are prepared to consider any views on revisions to the number, location and duration of meetings. You will note, for example, that our November meeting will be a full day meeting rather than the usual afternoon meeting.

Once a programme of events has been established, the Activities Officer assumes responsibility for briefing the speaker on what is expected and ensuring that the speaker remembers the engagement. We also need to ensure that the correct audiovisual equipment is available on the day. The Activities Officer has to liaise with the Secretary over room bookings and the Publicity Officer to ensure that members are told about meetings.

If any members would like to help out with the pre-planning of meetings, I would be delighted to hear from them — the actual commitment in terms of time is not high, and is often not much more than an informal get-together after a regular BAA meeting and a few phone calls.

### The 1993/4 BAA Committee

The elections for next year's committee will take place in time for the AGM on 11 June 1993. As in previous years, a number of current committee members have indicated that they are willing to continue in office. We are always keen to have as many nominations as possible for committee membership and this year we already have the prospect of at least one contested place.

I can confirm, therefore, that we will be having a postal ballot this year, so please send in lots more nominations!

The election timetable is:

10 May	final date for nominations
17 May	ballot papers posted
07 June	voting closes
11 June	results announced at the AGM

At the time of writing the following individuals have indicated their desire to stand for the Committee:

Chairman	David Eastwood (*)
Secretary	Duncan Pearson
Treasurer	Nicholas Small (*)
Journal Editor	Anthony Camacho
Activities	—
Education	Alan Mayer (*)
Technical	Jonathan Barman
Projects	George Macleod (*)
Publicity	Misha Jovanovic (*)
Recruitment	Richard Weber AND Mark Harris

(\*) — Holds this position in 1992/3

We would positively welcome further nominations, which must be proposed and seconded by current, paid-up members of the BAA. Candidates will be asked to provide me with a short statement for circulation with the voting papers.

We currently have two candidates for the post of Recruitment Officer: Richard Weber who teaches at Doncaster College of Further Education and currently helps on the Vector Working Group. Mark Harris runs Kestrel Consulting in partnership with Dick Bowman.

You will note that we are still looking for a nomination for activities officer and if anyone is contemplating standing for that (or any) post, they are welcome to contact me to discuss the work in more detail.

## Announcing "The APL Review"

A new and unique Journal for the International APL Community will be launched by the British APL Association in 1994. The APL Review will be fully refereed to the high standards required by the academic and research world, and edited by an international team of APL experts in a wide variety of fields.

Why a refereed journal? For many years now APL (and more recently J) has been employed by researchers who require more flexibility in their computing than is offered by other languages. For the result of their work to be added effectively to the accepted body of scientific knowledge two things are needed: (i) it must be "audited" (refereed) by their peers and (ii) it must be disseminated as widely as possible among those who may wish to use or add to the work. People who use APL professionally are well aware that they can learn a great deal from each other, even when their fields of study or application are very different. While an accountant is unlikely to read a software engineering journal, APLers in both disciplines will read an APL journal.

Why not Vector? A good question — Vector itself is one of the most widely read APL journals. But part of its appeal lies in its informal treatment of its subject matter. The more formal language and presentation of scientific "papers" sits awkwardly with the lighter tone of Vector, although many of the articles included in Vector over the years could have been presented as papers, and researchers have difficulty gaining credibility with peer review bodies when their articles are published in a journal alongside matter of a lighter nature. Vector will continue in its rôle, serving the whole of the APL community at all levels.

What about the annual International Conference proceedings? The APL Quote Quad volumes containing the papers presented at the annual conferences certainly play a vital rôle in disseminating the work of those who participate in the conferences, and we hope that they will continue to do so. Not everyone who

works with APL can attend the conferences, however, and not all work can be presented in a paper of suitable length to be included in the proceedings. More extensive topics often have to be presented as tutorial sessions, and for academics in particular this does not satisfy the need to have their work fully refereed and published. It is right that we should bring our work to the conference and share our experiences in this invaluable way, but for many areas of study a short paper for the purposes of conference presentation does not do justice to the work.

What sort of subjects will be covered? Virtually any original, serious application of APL or J. We will be looking for statistics, econometrics, engineering, actuarial applications, simulation, operational research and so on — all the areas where the language can be used to promote research and insight. At the same time, it will provide a vehicle for discussion of theoretical aspects of the languages themselves.

Who will read The APL Review? Everyone interested in APL. The wide subject coverage should ensure that there is something for everyone. Although by its nature it is a "serious" journal, this does not mean that the subject matter should necessarily be dry or purely technical. Journals are about communication, and The APL Review will publish papers that communicate at a level that is accessible to a large readership.

# News from Sustaining Members

*Compiled by Gill Smith*

## **Kestrel Consulting**

Kestrel Consulting combines the recruiting and entrepreneurial skills of Mark Harris (known to many British APL Association members from Kestrel Executive Selections) with the technical experience of Dick Bowman (former chairman of the British APL Association and presently a member of the ACM SIGAPL Executive Board).

Kestrel's objectives are two-fold.

The recruitment side of the company will continue to concentrate on specialist recruitment and placement of both permanent and contract staff, not only in APL but also in other focussed markets. Our aim is for our recruitment and placement services to operate globally. At the present time we have a very high demand for APL contract staff for the USA, and an equal if not greater demand for contractors with SAP experience. We intend to provide the very highest possible levels of service both to the individuals and to the employers.

On the technical side we view the introduction of GUI systems as an excellent opportunity for APL to re-establish itself as the high-productivity solution of preference. Our technical consultants have worked on hardware from the IBM PC right through to the largest mainframes, with operating systems from MSDOS to MVS/TSO, with software from Multimate to DB2, and on applications from accounting to zymurgy.

One of the means by which we aim to increase the amount of effective use of APL is widespread adoption of standardised tools and utilities; an initial step along this path is the Kestrel Software Library which is currently collecting software. If you have any software related to APL which you'd like to see widely distributed please contact us. Our emphasis is on wide distribution of low cost software; we are collecting now and plan to begin distribution in mid-1993.

Whether your needs are temporary hiring of maintenance programmers, custom development using an entire project team, consultancy and training, or any aspect of APL-related computing, Kestrel Consulting is only a telephone call, fax or email message away.



## Manugistics Inc

Here at Manugistics we've just come back from the U.S. Software Developers Conference, held every year just outside San Francisco. Bill Gates and Philippe Kahn were there, showing off their latest labour-saving devices for Windows programmers. We were there too, introducing a lot of 'C++' and Visual Basic fans to the concept of Array Programming. Now that APL\*PLUS works in Windows and talks to databases and other applications, we're turning a lot of heads — some "I haven't used APL for twenty years" folk, and quite a few "What is this APL stuff anyway?" types. It seems that even with all the promises ("Now you can build applications without writing a line of code!"), nobody else has a good solution to ad hoc problem-solving or complex data analysis. We are greatly encouraged by this state of affairs and will be pushing hard this year to open new doors and new minds to the power of APL.

Back in Rockville, we have just experienced a surge in sales as our year drew to a close at the end of February. APL\*PLUS II Version 5.0 was so successful that we ran out of our (anticipated) six months of stock less than three months after we started shipping. The release of APL\*PLUS/PC Version 11 was also responsible for a big boost as customers rushed to get their hands on the Runtime Binder so they can start sharing and selling their applications.

As you may be aware, the only way we could get APL\*PLUS II Version 5.0 out to the public was to lock the developers in a closet. The trouble with developing for Windows is that every neat new thing you add makes you realize there are three other things that would make it really, *really* neat. When you stay up all night to sneak those three in, there are, you guessed it, nine more the next afternoon when you get up. Well, as we expected, the developers broke out of the closet fairly quickly and have been working hard to get Version 5.1 ready. The development work is now nearly done and we are having a great time building 3-D forms with icons and bitmaps and graphic buttons and drawing pictures and using the vastly improved Wed form design tool and... ..and you'll have to wait for the rest. Actually, by the time you read this, we should be almost ready to start shipping!

In and around the work on Version 5.1, the team is working hard on preparing the next release of APL\*PLUS II for UNIX. Version 5 will include the APL2 features we have put into the 386 product and the JUser Command Processor as well as a lot of improvements and speed-ups. Release is planned for the summer.

With 5.1 finished, our Windows team can concentrate its design and development efforts on our "third generation" APL\*PLUS system, code-named INCA. This project is generating a great deal of discussion, white-board

wizardry, project journal writing and feature prototyping, but is still a way away from "Now Appearing on a Hard Disk Near You." As many of you will have heard or read, it will include Object-Oriented Extensions, Control Structures and an Alternative to Funny Characters. Stay tuned to this space for more details.

As you can see, we're doing everything we can to keep APL alive and growing well into the next century. Help us (and yourselves) by introducing (or re-introducing) your friends and colleagues to the new generation of APL!

### MicroAPL Ltd

On the APL front, much of our development work is aimed at the summer. We have, however, now completed the Macintosh version of our MicroPLOT business graphics software — PLOTMAC. Our MicroPLOT software is designed to facilitate the quick and easy production of business graphics. In common with all our GUI APLs, APL.68000 for the Macintosh includes an interface to the native graphics facilities but up to now the user will have had to write his or her own scaling routines to produce sensible, labelled, graphs.

The Quickdraw colour graphics available on the Macintosh offers the typical primitive graphics routines that one expects as standard — line drawing; shape drawing; pattern selection; fill routines; graphics text and so on. The MicroPLOT routines allow an APL programmer to set up a labelled graph via cover functions for the Mac Quickdraw routines. The PLOTMAC code is compatible with our MicroPLOT software for other systems such as the RISC System/6000 or for plotting devices such as Hewlett Packard plotters.

Registered Mac users will have received separate notification about the PLOTMAC software, but further details are available from MicroAPL.

Much of our consultancy work in recent months has been involved with producing Windows applications and we have scheduled another free seminar for the end of April 1993 entitled 'Developing Windows Applications'. This seminar is being jointly hosted by MicroAPL and Maddox Ford and will cover many of the practical problems encountered in designing and developing commercial Windows software.

In preparation for this seminar, we issued a GUI development questionnaire to our UK contacts. As you would imagine, the common factor that underlies our database is an interest in APL linked with a higher than usual number of non-PC owners (Mac, Atari, Amiga etc). Analysis of the results threw up some interesting statistics which we will be discussing in more detail at the seminar,

but as a preview, it is worth noting a few of the findings. Bear in mind that multiple answers are quite possible, so numbers don't add up to 100%

We asked people about their choice of GUI environment and came up with figures that seem to match the standard industry figures:

Windows	76%
Unix	21%
OS/2	10%
Mac	6%

The most favoured database was Oracle with 21% followed by Informix (8%) and Sybase (6%). The most popular database server was Novell (32%), followed by Unix (28%) then OS/2 (15%).

We asked about GUI development tools in two categories — 'low level' (such as C) or 'high level' (such as SQL Windows). For low level tools, C++ led with 26%, followed by C with 19% (APL clocked up 6%). The most favoured 'high level' tool was Visual Basic (19%). There was an even spread of names for other high level tools and APL re-appeared here as a possible high-level tool. Only one version of APL was mentioned.

## Dyadic Systems Ltd

All registered Dyalog APL/W users should by now have received their (free) upgrades to Version 6.3. The reception to 6.3 during the Beta Test was extremely positive and judging from the efforts of some users commercial APL-based products — that are indistinguishable in terms of look and feel from other Windows applications — should soon begin to appear.

The new `□NA` facility to call Windows DLL routines is proving to be exceptionally popular, not least because it is entirely dynamic and very fast. With `□NA`, users have themselves been able to generate powerful and efficient interfaces to various other products including Oracle and Microsoft SQL Server. Insight Systems in Denmark is even using `□NA` to provide a facility that will connect Dyalog APL/W users to practically any commercial database running on almost any host system — directly from a Windows workstation. Contact Morten Kromberg on +45-4210-7022 for details.

The device-independent graphics in Version 6.3 has also been well-received, as has the fact that you can at last produce APL listings on practically any printer with no effort at all. For this, Dyadic is grateful to Adrian Smith for allowing the distribution of his excellent TrueType APL font with Dyalog APL. The new

version of *WSDOC*, which is included with 6.3, has a Windows front-end, and makes the production of attractive looking APL listings a very simple matter indeed.

To illustrate the powerful new bitmap-handling capabilities, Dyalog APL/W includes a version of the "Arachnid" card game (a form of Patience) which is coded entirely in APL. This game requires a certain amount of skill, and none of the beta-testers discovered the APL coding error that crashed the workspace when the game was successfully completed! On a reasonably fast PC, the APL workspace is practically identical in look and feel to a standard Windows card game, and it amply demonstrates that Dyalog APL/W can compete effectively with other GUI development tools. The new version of the *WDESIGN* workspace takes full advantage of the graphical facilities built in to the interpreter. New objects are created by clicking on a "Visual Basic look-alike" toolbar (in place of the buttons used in version 1), and the workspace now includes interactive facilities to design icons, cursors, bitmaps, and your own custom "toolbars". *WDESIGN* supports all the new object types provided in Version 6.3, so you can also use graphics to add 3-D effects to your dialog boxes.

On the Unix front, Dyadic has produced new releases of Dyalog APL for the IBM RS/6000 (AIX 3.2) and for Sun Workstations (Solaris 2.1). The RS/6000 version is up to 10% faster than previous releases. Work on a Motif implementation, based on Dyalog APL/W, has begun in earnest.

Dyadic is pleased to announce the recruitment of Graeme Robertson, ex IPSA, who joins the team as an APL consultant. The company is also delighted to report its most successful year of trading since it was established in 1979.

# THE EDUCATION VECTOR

April 1993

*Editor Alan Mayer*

This Education Vector has been reprinted from VECTOR Vol.9 No.4. VECTOR is the Quarterly Journal of the British APL Association. For more information about the British APL Association, please contact: Anthony Camacho, 11 Auburn Rd, Redland, BRISTOL, BS6 6LS Tel: 0272-730036.

## Contents

Editorial	Alan Mayer	28
A Beginner's Guide to Low-Cost 'Real-Work' APLs	Dick Holt	29
Workshop: Learning Mathematics with APL	Howard Peelle	36
An APL Scrabble Bag	Bill McLean, Ted Emms	41

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

Ysgol Rheolaeth Busnes Ewropeaidd  
Parc Singleton,  
Abertawe SA2 8PP

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

## Editorial

Hello, and welcome to your "Spring" copy of Education Vector. This is written with a certain amount of sarcasm, as I gaze out over a wind-swept and rain-soaked Swansea. But I trust that by the time you read it Spring will indeed have come and the sun will be shining (you see, I am an eternal optimist at heart).

Last time, I promised you a detailed comparison of the low-cost (and in some cases free) APL interpreters. The article by Dick Holt in this edition, which has been circulated in various forms via various media, is an attempt at a "beginner's guide". It originates from the APLSIG of the Capital PC User Group, who invite everyone to circulate/republish it as widely as possible. As always, I shall be pleased to send copies of this edition and future editions of Education Vector to anyone who sends me their address.

While I am grateful for the work that has been done to make this comparison between interpreters, I feel that there is more that could be done, perhaps with contributions from our own readers. The report as it stands is largely aimed at business applications, which do not always share the requirements of educational establishments. For example, I-APL may be slower and offer less memory than its rivals, but the fact that it is available for a wide variety of machines, many of which are currently being used in our schools, makes it an attractive option for educational purposes.

For many, of course, the low-cost interpreters provide an easy path into the use of APL, whether for business, educational or other purposes. It is clearly part of our job to encourage this growth in the applications of APL. I should point out, however, that I have yet to find a low-cost interpreter that is a serious rival to those produced for professional purposes. For my own work I use Dyalog and APL\*PLUS, and I have colleagues using APL.68000 and APL2 — none of us is about to give up the professional environment for a low-cost option. That having been said — I will be pleased to hear from anyone with contributions to make to the discussion, especially those who have used one or more of the packages in the educational environment.

The promised follow-up to Howard Peelle's article on Workshop Design is a Sample Topic on "Triangular Numbers". I find the method of presentation very instructive, and I hope it helps and encourages others to produce material of similar high standard.

Finally, all you APLers out there — drop me a line to tell me about your latest APL project. I am always looking for material that may be of interest to our readers. And if you are not already members, why not think about joining the British APL Association? If you do not have an application form, write to Rowena Small, 8 Cardigan Road, London, E3 5HU. UK subscription is currently £12 (students £6).

# A Beginner's Guide to Low-Cost "Real-Work" APLs

by Dick Holt

This article is a beginner's guide to low-cost APL interpreters, compiled by the APLSIG of the Capital PC User Group (CPCUG). Comparisons emphasize work and business applications — because it's the workplace that generates demand for technological innovation, and pays for it. I emphasize free or low cost APLs, because these are likely entry points for exactly the kind of people who are most badly needed by the APL community — namely newcomers.

It's an "apples and oranges" comparison because the various APLs are themselves "apples and oranges". It may contain errors of fact, as well as opinions that are wrongly informed. Corrections are invited. [I will be pleased to receive, pass on and publish any comments/corrections — Ed.]

A central theme of this article is that APLs vary widely in size, speed, performance, and price. An advantage of this heterogeneity is that APL offers "something for everyone". A disadvantage is that there is little compatibility among versions. In any event, at least two free or low cost APLs — with big workspaces, good speed, and many useful features — should now be increasingly attractive to people who have never used APL before.

To begin, here are a few speed and size comparisons, referenced to the free demo of APL\*PLUS v10.1, a widely used commercial APL:

**Table 1: Size, and Relative Speed: For five simple benchmark tests (B1-B5).**

Interpreter	B1	B2	B3	B4	B5	□WA * CLEAR WS
APL*PLUS demo	>1.00	>1.00	1.00	1.00	1.00	0
APLI386	1.13	2.11	>0.24	0.97	>0.03	1044k
APLIPC	1.11	2.36	0.50	1.25	0.12	262k
APLISW	1.39	2.36	0.50	1.25	0.12	312k
TryAPL2	4.52	2.48	4.17	>0.36	>0.03	233k
Sharp APL/PC	1.46	2.41	0.67	1.40	0.15	195k
I-APL	20.78	30.95	26.50	10.22	79.40	31k

All benchmarks times are normalized to APL\*PLUS demo = 1.00

*Benchmark B1:  $Q \times 9$ ; where  $Q$  is  $(2.5 + i1800)$ : 1800 exponentiations.*  
*Benchmark B2:  $(i3000) \div 44.3$ : 3000 floating point divisions.*  
*Benchmark B3:  $+ / i15000$ : 15000 integer additions.*  
*Benchmark B4:  $20!40$ , looped 1000 times (no, looping doesn't cost all that much time).*  
*Benchmark B5: length 10,000 boolean vectors, and-ed and or-ed 10 times (100,000 boolean operations).*

- \* Defaults for my machine:  $\square WA$  for APLI386 is as large as RAM permits, and may be increased further by a user-transparent virtual (on-disk) workspace.  $\square WA$  for APL\*PLUS demo is zero, since the )CLEAR WS is, by design, inaccessible.  $\square WA$  for others is configuration dependent, except for I-APL, which is limited to 31.6k on DOS machines (less if cover functions are used for graphics, files, help, etc.).

Benchmarks are designed to avoid *WS FULL* in I-APL, and to run in a few minutes or less on my machine. Times are normalized to APL\*PLUS, and are independent of disk speed.

## Observations

Speed and size vary enormously. APLI386 has a  $\square WA$  30 times larger than I-APL (on my machine), and does integer addition 17 times faster than TryAPL2. For exponentiation, TryAPL2 is 3-4 times slower than all except I-APL. All Iverson software is 2-4 times faster than APL\*PLUS for  $+ / i15000$ . APLI386 is more than 100 times faster than I-APL for  $+ / i15000$ .

For dyadic shriek (!), TryAPL2 is 3 times faster than both APL\*PLUS and APLI386, and 29 times faster than I-APL. Although not commonly used, dyadic ! is purposely chosen to highlight speed variations. Non-integer arguments for ! weren't tested.

For boolean operations — a putative strong point of APL — APLI386 and TryAPL2 are 30 times faster than APL\*PLUS, and more than 2500 times faster than I-APL. Both APLI386 and TryAPL do boolean operations so fast that I had to run them  $10E7$  times to find out which was faster. TryAPL2 seems to be about half of one percent faster than APLI386.

These speed tests aren't directly comparable: Iverson APLI386 uses the 32-bit instruction set of the 386 chip — none of the others do. Benchmarks can be notoriously misleading, and your times may be different. Run your own tests.

Blitzing speed and big workspaces are important, but they're not the only things to consider. Effective use of APL also depends on features like a full-screen



editor, vendor support, useful application development tools, good documentation, and smooth upgrade paths.

Hardware is also a factor. Only APLI386 requires a 386 machine. All others work on a 386 and on lesser machines. Among free or low-cost interpreters, only TryAPL2 requires a colour monitor (EGA or better), but Iverson's APL characters are inaccurate on a monochrome monitor.

Can I do "real work" in any of these low-cost APLs?

- APL\*PLUS demo: No — User-defined functions/system commands inaccessible
- TryAPL2 : Yes — But with del editor, no DOS or APL files
- APLI386 : Yes — 386 required, also does Windows
- APLIPC : Yes — 386 not required
- Sharp APL/PC : Yes — Limited: poor documentation, not easily available
- APLISW : ? — Limited: no DOS files, no APs
- I-APL : ? — 31k WS, slow, no error trapping

To be fair, the APL\*PLUS demo and I-APL aren't designed to do real work. They're designed to showcase product features or to teach APL. Within varying limits, real work is feasible with all other APLs.

Table 2 Upgrade Path and Prices (\$US)

Interpreter	Type	Disk Price	Documentation Price ***	\$ Full Product	\$ 386 Version
APL*PLUS demo	flat	free*	free/on screen	\$695	\$1700
TryAPL2	nested	free*	free LJ print	\$500	Included
APLI386	boxed	\$30	\$30 2 books	Is full product	
APLIPC	boxed	\$30	\$30 2 books	APLI386	APLI386
APLISW	boxed	free	some on screen	APLIPC	APLI386
Sharp APL/PC	boxed	free	unavailable?	APLIPC	APLI386
I-APL	flat	free	\$25	Any of the above	

\* Free from original source (see "Sources" below), and downloadable free from the BBS\APL. Request to download may be needed. Some suppliers may charge a nominal fee.

\*\* Disks, but not documentation, may be downloaded free from the BBS\APL, and may also be available elsewhere.

\*\*\* Documentation for Sharp APL/PC may be hard to get and is hard to use. Iverson documentation is an imperfect hybrid of Sharp APL/PC (orphaned v17) material, mainframe material, and new on-screen Iverson material.

Prices may be discounted: street price, volume, educational, dealers, etc. Ask. See "Sources" below.

IBM (TryAPL2 and APL2/PC), Iverson (and Sharp), and APL\*PLUS II (386 version) all provide “nested” or “boxed” arrays — all in incompatible ways. Also, many “flat” features aren’t compatible among various APLs, and APL\*PLUS isn’t fully compatible with APL\*PLUS II.

Because of numerous and deep incompatibilities, switching from one APL to another can be an obstacle to upgrading.

### Keyboard and Other Incompatibilities

Keyboards for IBM and APL\*PLUS are similar. They’re the traditional APL keyboards. Iverson keyboards are slightly different, but resemble the APL\*PLUS “unified” keyboard as used in the APL\*PLUS demo. Many Iverson APL keys are in the “traditional” place, differing mainly in the use of lower case, and in the use of <ALT> rather than <SHIFT> to get special APL characters. Most Sharp keys are in the same place as Iverson keys. Users of these APLs can probably switch from one to another without too much difficulty, at least as far as the keyboard is concerned. Non-keyboard incompatibilities — mainly in quad functions and shared variables — are also difficult. TryAPL2, APL\*PLUS demo, Iverson Software, and CPCUG APL lessons all have on-screen keyboard diagrams and on-screen help.

Because I-APL attempts to adhere to ASCII, its keyboard is almost completely unrelated to any other APL keyboard. Does this mean that the I-APL keyboard is “bad”? No. It’s simply that switching from I-APL to any other more powerful APL is harder than switching among these other APLs. From a beginner’s perspective, all APL keyboards are “bad”. In its 1991 APL classes, the CPCUG found the APL keyboard to be the greatest single obstacle to learning APL. Students who begin with I-APL must learn a new APL keyboard if they want to upgrade to a more powerful APL.

### Editors

A full-screen editor is essential for modern programming, in APL or in any other language. Among commercial products, APL\*PLUS and APL\*PLUS II have a superior built-in assembly language editor. Iverson products have a full-screen editor function that may be copied into your WS. IBM’s APL2/PC also has an editor function that may be copied into your WS, or you may also use something like IBM’s “Personal Editor” for APL2/PC. TryAPL2, Sharp APL/PC, APLISW, and I-APL have only the obsolete “del” editor. The del editor is a line editor like DOS’s EDLIN, lacking only most of its convenience.

**Full-Screen Management**

- APL\*PLUS v10 — Yes, excellent, versatile, strong
- APL\*PLUS demo — No, screen management is user-inaccessible
- TryAPL2 — Yes, uses shared variables
- IBM APL2/PC — Yes, uses shared variables
- APLI386 — Yes, uses shared variables
- APLIPC — Yes, same as APLI386 above
- Sharp APL/PC — Yes, difficult
- I-APL — Yes, has FSCREEN.IWS with GET and PUT cover functions
- APLISW — No

<b>Files</b>	<b>DOS Files</b>	<b>APL Component files</b>
APL*PLUS demo	No	No, quad functions inaccessible
APL*PLUS	Yes	Yes
IBM APL2/PC	Yes	Yes
APLI386	Yes	Yes
APLIPC	Yes	Yes
Sharp APL/PC	Yes	Yes
I-APL	Yes (cover fns)	No (cover fns available)
TryAPL2	No	No
APLISW	No	No

**Documentation**

- APL\*PLUS demo — On-screen keyboard tutorial/description of features.
- TryAPL2 — On-screen doc. & lessons; LJ printable user manual.
- APLI386 — On-screen, plus two inexact books.
- APLIPC — Identical to APLI386 above.
- APLISW — Some, on-screen.
- I-APL — Good on paper, not much on-screen.
- Sharp APL/PC — None on-screen, paper not easily available, difficult.

**On-screen Help Supplied**

- APL\*PLUS demo — Excellent, also excellent for full products.
- TryAPL2 — Good, in WS TRYDOC.TRY.
- IBM APL2/PC — I don't know.
- APLI386 — F1: Fairly complete, paper documentation essential.
- APLIPC — F1: Identical to APLI386 above.
- APLISW — F1: Identical to APLI386 above.
- Sharp APL/PC — No.
- I-APL — No.

**Function Keys:** settable on all except APLISW and I-APL. Varying degrees of pain on all others.

**APL Lessons Availability:** CPCUG APLSIG's generic, interactive, on-screen lessons are available for all APLs discussed here, some also in French.

**Non-English Language Keyboard/Character Support:** This feature is important only if you care about being competitive in today's global marketplace (not everybody does). Among APLs discussed here, only TryAPL2 includes instructions for the keyboard and National Code Pages. On-screen non-English characters and non-English keyboards can be made in other APLs, with varying degrees of pain.

#### **Calls to Machine Language/C/Fortran etc.**

APL*PLUS	— Yes
APL*PLUS demo	— Yes, example provided of calls to C
TryAPL2	— No, requires AP210, not included in demo
IBM APL2	— Yes, FORTRAN WS included in product
APLI386	— <input type="checkbox"/> NA type improvements planned
APLIPC	— <input type="checkbox"/> NA type improvements planned
I-APL	— Yes, machine language expertise needed
APLISW	— No

#### **Commercially Motivated**

<b>Vendor Support</b>		<b>Kaizen Strategy?*</b>	<b>Market Responsive?</b>
APL*PLUS	— Hot-line + bbs	Yes, to a fault	Highly so
APL*PLUS demo	— Ask salesperson	Apparently	Very Likely
TryAPL2	— Hot-line, free	Yes	Apparently
IBM APL2	— Hot-line, free	Yes, not to a fault	Unknown
APLI386	— Hot-line, toll	Yes	Apparently
APLIPC	— Hot-line, toll	Yes	Apparently
APLISW	— No	No	No
I-APL	— No	No	No

\* Kaizen is the Japanese term for a reliable corporate commitment to unceasing, market-driven, improvement and innovation.

**Run-Time Systems:** Programming APL for run-time systems, whether for commercial use or just for your work group, requires strong skills in error trapping. All APLs here, with the exception of I-APL, have error trapping features — and all are incompatibly different. TryAPL2's error trapping is undocumented.

IBM APL2 includes a free run-time .EXE system. APL\*PLUS and APL\*PLUS II provide run-time systems at varying prices: APL\*PLUS 5-pack is \$250, unlimited is \$995; APL\*PLUS II 5-pack is \$875, unlimited is \$5000.

Pricing strategy for Iverson APLI386 and APLIPC make run-time systems inexpensive for distribution in small numbers. If you develop an application that will sell many copies, APL\*PLUS may be cheaper. I-APL may be distributed free, but commercial use is prohibited.

**Odds and Ends:** Iverson Software products (and Sharp) have built-in complex numbers. However, complex number cover functions are not hard to write, and are available as shareware, or come with some other APLs. Only I-APL uses "direct definition" functions. Only APL\*PLUS II and Iverson APLI386 do windows. Only I-APL is available on a very wide range of machines, such as ATARI, BBC, Archimedes, or Mac II.

Other Info	Graphics	Color	Printer Support
APL*PLUS	Yes	Yes	V. good
APL*PLUS demo	Yes	Yes	not accessible
IBM TryAPL2	Yes	Yes	V. good for LJ
IBM APL2/PC	Yes	Yes	V. good, via AP80
APLI386	Yes	Yes	V. good (LJ, PS, DM)
APLIPC	Yes	Yes	Identical to APLI386
I-APL	Possible	Possible	Limited
APLISW	No	No	No

## Sources

APLI386/APLIPC: Iverson Software Inc.  
33 Major Street, Suite 466  
Toronto, Ontario Canada M5S 2K9  
Tel: 416-925-6096, Fax 416-488-7559

TryAPL2: IBM APL Development (specify disk size)  
M46/D12-278B Santa Teresa Lab  
Box 49023, San Jose CA 95161-9023 USA

APL\*PLUS demo: Manugistics Inc.  
2115 East Jefferson St. Suite 729  
Rockville MD 20852 USA 301-984-5412

I-APL:		
USA	<b>UK inquiries</b>	<b>UK Orders</b>
6611 Linville Dr.	11 Auburn Rd. Dept. 278	59 The Crescent,
SQ5 Suite 314	Redland	Milton, Weston-super-mare
Weed CA 96094	Bristol BS6 6LS	Avon BS22 8DU

## Call for Action

This comparison of APLs is a collaborative effort by the APLSIG of the CPCUG. We've invited corrections and improvements from APL vendors. I'm not sure that we've got everything right. Readers are invited to send corrections and improvements to the editor of Education Vector, who will forward them to the author.

# Workshop: Learning Maths with APL

by Howard Peelle, University of Massachusetts

## Sample Topic: "Triangular Numbers"

First, the primitive function  $\uparrow$  is introduced by an example:

```

      17
    1 2 3 4 5 6 7
  
```

The example is intended to be strongly suggestive in provoking thinking. The learner may well try other examples, such as  $\uparrow 6$  or  $\uparrow 8$  or even  $\uparrow 1000$  (which would overflow the screen while instilling a certain sense of power) or better yet *experiment* systematically ( $\uparrow 5$ , then  $\uparrow 4$ , then  $\uparrow 3$ , then  $\uparrow 2$ , then  $\uparrow 1$ ) to see a pattern clearly. Then one can *infer* how to define the function precisely:  $\uparrow$  is the Integer function, which generates all consecutive indices starting with 1 up to (and including) the integer given. The learner might also test special cases, such as  $\uparrow 0$  (which produces an empty list, yet is consistent with the above definition) and discover the limits of its domain — ruling out negative numbers and decimal numbers (which yield *DOMAIN ERROR* from the computer).

Further, the syntax is important to notice:  $\uparrow$  is a "monadic" function (i.e. it has one input), in contrast to "dyadic" functions, such as  $+$  and  $\wedge$  (which have two inputs). This distinction may have been introduced already, perhaps along with the *new knowledge* that all APL functions are either monadic or dyadic. Nevertheless, this is an opportunity to reinforce this important APL concept while monitoring *disequilibrium* with the learner's previous understanding of function syntax, which may need clarification due to inconsistencies inherent in conventional mathematical notation (see [1]).

Careful *interpretation* is also needed because the learner may well try using  $\uparrow$  with other functions, as in  $10 \times \uparrow 7$ , which gives 10 20 30 40 50 60 70; or  $\uparrow 7 \times 10$ , which yields 1 2 3 ... 69 70. The contrast is important to note — 10 Times the result of  $\uparrow 7$  vs.  $\uparrow$  of the result of 7 Times 10 — because it raises the issue of how APL evaluates expressions with more than one function. Through experimentation, the learner can realize APL's rule: every function uses the entire expression on its right (unless indicated otherwise with parentheses). This rule applies equally to all functions; that is, there is no hierarchy of functions in APL, as there is in conventional mathematics (exponentiation before  $\times$  and  $/$  before  $+$  and  $-$ ).

Although APL appears to accommodate many cases (e.g.  $(2 \times 3) + 4$  is 10), it can produce some unexpected results (e.g.  $2 \times 3 + 4$  is 14 in APL because  $3 + 4$  is done first). While it may take more time for the learner to adjust to APL, at this stage it is advisable to use redundant parentheses to clarify order of operation. Eventually changing to this new rule can *empower* the learner by offering some 60 primitive functions on the keyboard (plus more which can be derived or defined as programs) and the convenience of a single, completely general rule to handle any expression with any number of functions, without having to memorize or look up which gets done before another.

The next expression shown on the handout is:

$+ / 17$	<i>The sum of the integers from 1 to 7</i>
28	is 28

The Sum function  $+ /$  has probably been introduced already (as in the topic on Averaging). If not, for now suffice it to say that  $+ /$  inserts  $+$  between each number (and evaluates the resulting expression). And defer interest in exploring the operator  $/$  (which can be used with other functions) for another time.

The overall objective of this topic is to explore *alternative* ways of finding the sum of consecutive integers. Accordingly, the next four expressions illustrate a different algorithm.

$\phi 17$	<i>The reversal of the integers 1 to 7</i>
7 6 5 4 3 2 1	

The learner may explore the behaviour of the Reverse function  $\phi$  on other lists (including characters, for instance  $\phi$  'LIVE' is 'EVIL') or indeed on other arrays such as matrices (which become reversed about the vertical axis, as indicated mnemonically by the symbol  $\phi$  itself). This leads naturally to another topic: palindromes (not discussed here).

$(\phi 17) + (17)$	<i>The result above added to integers 1 to 7</i>
8 8 8 8 8 8 8	

APL performs this addition in parallel, automatically. Learners may try similar expressions with other appropriate functions. Since the concept of parallel processing is likely to be new knowledge, it may need to be addressed again later (as in a "spiral curriculum"), perhaps including the special case of a single value (e.g.  $10 \times 17$  mentioned above). For now, however, the learner should concentrate on building an expression to *represent* this particular algorithm, which involves adding a list to itself. Incidentally, another similar algorithm

folds the list about its centre point but raises the issue of odd vs. even length lists (not discussed here).

$+/(\phi 17)+(17)$       *The sum of the list (of seven 8s) above*  
 56

Note that  $+/$  applies to the result of everything to its right.  $(+/(\phi 17)+(17))\div 2$  Half of that is 28 28. The learner may notice that this is the same result obtained earlier (by summing directly using  $+/$ ). So, this is apparently another way to find the sum of counting integers!

This algorithm can be embodied in a "glass box" program for *further study*, like this:

*GAUSS: (+/(\phi 1\omega)+(1\omega))\div 2*

The name of the program (*GAUSS*) is chosen for historical reasons (see story below).

Note use of  $\omega$  to represent the input, which can be any integer. If other mechanics of defining a program have not been introduced yet, just say: "Type a name followed by a colon, then the algorithm (expressed in terms of  $\omega$  for the input it will get later)" and be prepared to help in subsequent editing.

Giving an algorithm a name provides a more convenient way to try it out. For example:

*GAUSS 7*      *Using program GAUSS with an input of 7*  
 28

The learner may try it again for a different input (besides 7). *Reflecting* on the algorithm can lead to insight: there are always  $N$  copies of  $N+1$  to be summed and then halved.

Anecdotally, there is a story about the young German mathematics prodigy, Karl Friedrich Gauss, who is said to have discovered a simple way of calculating the sum of  $N$  integers in the late 18th century. When asked by his schoolmaster to take out his slate and add the integers from 1 to 1000 (or some such large number) in order to keep Karl busy for a while (since he was probably bored and disturbing other students and the teacher with difficult questions), Karl came up with the answer startlingly fast. Here, having experienced the development of the algorithm (above), the learner can join in speculation about how Karl might have done it.



Finally, another expression is given. It also results in 28:

$$7 \times (7+1) \div 2$$

28

This may be recognised as a particular case of what has come to be known as "Gauss's Formula":  $n(n+1)/2$  in conventional notation. The formula calculates the sum of the integers from 1 to n. It may be compared with the algorithms above and seen to be exceedingly more efficient. The learner can easily express this in APL:  $N \times (N+1) \div 2$  and try it for any value(s) of N or, better yet, *construct* another glass box program:

```
TRI: w*(w+1)÷2      Program TRI (a suggestive name)
```

This program may be used with several input values at once, yielding *patterns* worthy of further study. For example:

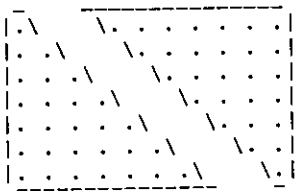
```
TRI 10 100 1000 10000 100000
55 5050 500500 50005000 500000500000
TRI 1 2 3 4 5 6 7 8 9 10
1 3 6 10 15 21 28 36 45 55
```

Note: the 7th TRI result is 28.

These are "triangular numbers", so named because they represent the number of (lattice) points in a triangle, as shown below:

<pre> .                 </pre>	<p>Note: The first triangular number is 1 (one dot in the first row); the second is 3 (1+2 dots in the first two rows); third is 6 (1+2+3 dots in three rows); fourth is 10 (1+2+3+4 from four rows); fifth is 15; sixth is 21; and seventh is 28 (1+2+3+4+5+6+7 from all seven rows).</p>
--	--

Thinking of a triangular number as an area leads to a nice geometric representation of Gauss' formula as half the area of a N by N+1 rectangle formed by sliding two triangles together:



Finally, the learner may now be informed that APL provides a primitive function, called Sum-Scan, which produces a list of triangular numbers directly:

```
+ \ 1 7
1 3 6 10 15 21 28
```

Exploring this function by itself may lead to *more discoveries*, for instance, that Sum-Scan of odd integers produces squares:

```
+ \ 1 3 5 7 9
1 4 9 16 25
```

And successive Sum-Scans are found in Pascal's triangle (which is a likely next topic for a workshop).

For now, in closing, the learner can put *GAUSS* and *TRI* in his/her conceptual "tool box" (as well as save them on a computer disk) for later modification and use, perhaps along with other tools, for learning mathematics.

## Reference

- [1] Peelle, H.A. "Introduction to APL for Learning Mathematics", Journal of Computers in Mathematics and Science Teaching, Vol.9(1) Fall 1989.

## Sample Hand-out Worksheet: Teaching Mathematics with APL

Examine the following APL expressions and write annotations in the margins on this page (and back, if needed) briefly describing what you think each expression does.

17	+ / 17
$\phi$ 17	( $\phi$ 17) + (17)
+ / ( $\phi$ 17) + (17)	(+ / ( $\phi$ 17) + (17)) $\div$ 2
7 * (7 + 1) $\div$ 2	

As you proceed, use the computer to explore unfamiliar symbols by trying additional similar expressions. By observing patterns in the results, try to infer how each symbol works in general.

Review the overall sequence of expressions given and identify the algorithm or topic illustrated here. Embody the algorithm in a program, modify it for your own purposes, explore related topics (if you have time), and mention possible applications of the program(s).

# An APL Scrabble Bag

*Contributed by Bill McLean and Ted Emms*

This simple game is a nice little exercise in APL. 100 Scrabble tiles (with letters in various proportions) are mixed up and drawn in groups of five. You take the five letters presented, plus two optional letters of your own choosing, and try to make an anagram. If you succeed, you score the value of the tiles drawn. The computer keeps the score.

```
[0] BAG;ALPH;COUNT;LETTERS;ORDER;PLAYERB;SUM;BLOCK;I;NUMS;PLAYERA;SCORE;[]IO
[1] LETTERS←'ABCDEFGHIJKLMNPOQRSTUVWXYZ[]'
[2] SCORE←1 3 3 2 1 4 2 4 1 8 5 1 3 1 1 3 10 1 1 1 1 4 4 8 4 10 0
[3] NUMS←9 2 2 4 12 2 3 2 9 1 1 4 2 6 8 2 1 6 4 6 4 2 2 1 2 1 2
[4] ALPH←NUMS/LETTERS
[5] PLAYERA←0
[6] PLAYERB←0
[7] SUM←0
[8] I←0
[9] []IO←0
[10] ORDER←100?100
[11] BLOCK←20 5p(ALPH[ORDER])
[12] LINE←[]←BLOCK[I;]
[13] COUNT←LETTERS\BLOCK[I;]
[14] SUM←+/SCORE[COUNT]
[15] →(2|(I,I+1))/EVEN,ODD
[16] ODD:'PLAYER A. DID YOU MAKE AN ANAGRAM? (Y/N)'
[17] →('YN'=1+[])/AA,BB
[18] AA:PLAYERA+PLAYERA+SUM
[19] BB:I+I+1
[20] ←LINE
[21] EVEN:'PLAYER B. DID YOU MAKE AN ANAGRAM? (Y/N)'
[22] →('YN'=1+[])/CC,DD
[23] CC:PLAYERB+PLAYERB+SUM
[24] DD:I+I+1
[25] →(I<20)/LINE
[26] 'PLAYER A: SCORE = ',▼PLAYERA
[27] 'PLAYER B: SCORE = ',▼PLAYERB
```

The scores for the different letters are set up in line 2, the numbers of each letter in line 3, and the random ordering is set up in line 10, and arranged in rows of five in line 11. Inside the loop (lines 12-25) the score is calculated in lines 13 and 14. Line 15 determines whose turn it is. The scores are only added if an anagram is claimed.

A typical "game" goes something like this:

```
BAG
IGETQ
PLAYER A. DID YOU MAKE AN ANAGRAM? (Y/N)
N
EOFOV
PLAYER B. DID YOU MAKE AN ANAGRAM? (Y/N)
N
TIEPT
PLAYER A. DID YOU MAKE AN ANAGRAM? (Y/N)
Y
SRENS
PLAYER B. DID YOU MAKE AN ANAGRAM? (Y/N)
Y ...
```

and so on, until you have had ten goes each...

```
AIHEM
PLAYER B. DID YOU MAKE AN ANAGRAM? (Y/N)
Y

PLAYER A: SCORE = 56
PLAYER B: SCORE = 42
```

... and you can start all over again!

# APL Product Guide

*Compiled by Gill Smith*

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

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

We reserve the right to edit material supplied for reasons of space or to ensure a fair market coverage. The listings are not restricted to UK companies and international suppliers are welcome to take advantage of these pages.

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

- Complete APL Systems (Hardware & Software)
- APL Interpreters
- APL Visual Display Units and Printers
- APL-based Packages
- APL Consultancy
- Other Products
- Overseas Associations
- Vendor Addresses

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

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

All contributions and updates to the APL Product Guide should be sent to Gill Smith, at Brook House, Gilling East, York, YO6 4JJ. Tel: 04393-385

## COMPLETE APL SYSTEMS

COMPANY	PRODUCT	PRICES (£)	DETAILS
Active Workspace	AWL486	1,950	486 based 33MHz PC, 140MB Disk, 4MB RAM, VGA Colour. (Inc. 1 year on site maintenance.)
APL People	IBM PCs & compatibles	poa	Includes PC, mono/colour monitor, APL Interpreter, operating system software, plus optional printers, graphics boards, additional memory etc.
Dyadic	IBM RS/6000 MD320	11,796	APL POWERstation (Greyscale) 27.5 MIPS, 7.4 Mflops RISC Processor 8Mb RAM, 120Mb Disk 19" 1280x1024 Greyscale Graph Display AIX, OSF Motif, Dyalog APL (1-user)
	IBM RS/6000 MD320	13,817	APL POWERstation (Colour) 27.5 MIPS, 7.4 Mflops RISC Processor 8Mb RAM, 120Mb Disk 16" 1280x1024 Colour Graphics Display AIX, OSF Motif, Dyalog APL (1-user)
	IBM RS/6000 MD320	22,656	Advanced APL POWERstation 27.5 MIPS, 7.4 Mflops RISC Processor 16Mb RAM, 320Mb Disk, 150Mb Tape 16" 1280x1024 Colour Graphics Display AIX, OSF Motif, Dyalog APL (1-user)
	IBM RS/6000 MD520	37,114	APL POWERsystem (8-users) 27.5 MIPS, 7.4 Mflops RISC Processor 16Mb RAM, 320Mb Disk, 150Mb Tape CD-ROM Drive, 16 Ports AIX, Dyalog APL (2-8 user licence)
	IBM RS/6000 MD530	72,054	APL POWERsystem (16-users) 34.5 MIPS, 10.9 Mflops RISC Processor 32Mb RAM, 1.34Gb Disk, 2.3Gb Tape CD-ROM Drive, 16 Ports AIX, Dyalog APL (8+ user licence)
	IBM RS/6000 MD540	122,842	APL POWERsystem (32-users) 41 MIPS, 13 Mflops RISC Processor 64Mb RAM, 1.7Gb Disk, 2.3Gb Tape CD-ROM Drive, 32 Ports AIX, Dyalog APL (8+ user licence)
Interprocess Systems	APL2 Dev't Workstation	poa	Mainframe APL2 supported on a PS/2 via a co-processor card with 16Mb of memory running VM/ESA (370 mode). A complete system includes a PS/2, a P/370 co-processor card, and software licenses for VM/ESA, APL2, GDDM and the full line of interprocess APL2 enhancements.
MicroAPL	IBM RS/6000	12,000+	POWER range of RISC systems running AIX. Dumb terminal or graphical interface.
	Aurora	20,000+	Multi-user APL computer using 68020 CPU. Std. configuration 2Mb RAM, 16 RS232 ports, 68 Mb hard disc, 720K diskette
Optima	IBM Compatible	poa	Complete PC-based station, APL Interpreters & all support eq't

## APL INTERPRETERS

COMPANY	PRODUCT	PRICES (£)	DETAILS
Active Workspace	DYALOG APL DOS 386	poa	Dyadic's PC 386 APL Interpreter.
APL Software	APL*Plus/PC Release 10	450	STSC's APL for IBM PCs & compatibles. Upgrades from earlier releases also available.
	Run-time	poa	Closed version of APL*Plus/PC which prevents user exposure to APL
	APL*Plus II	1,395	All the features of mainframe APL*Plus for your 386PC!
	Run-time Dyalog APL	poa 1000-10,000	2nd generation APL for Unix systems
	APL2/PC	poa	IBM's APL 2 for the PC.

Cocking/Drury	APL*PLUS PC Rel 10	410	STSC's full featured APL for IBM's and compatibles - Version 10 includes the Quad-NA facility to interface to non-APL software, support for Microsoft Windows and mouse devices. The User-command processor has been built in to the Interpreter.  Upgrades to version 10 are available from Version 9 and earlier releases.
	APL*PLUS PC Run-Time	175 for 5	Closed version of the Interpreter for developers, prevents user exposure to APL.
	APL*PLUS PC Developer System	950	Gives rights to distribute an unlimited number of copies of Run-Time application.
	APL*PLUS II System	1200	High powered APL Interpreter for the 80386 chip. Price Includes one year's maintenance and free upgrades - volume discounts VERSION 4.0 NOW AVAILABLE - THE ONLY APL THAT LETS YOU BUILD WINDOWS & DOS APPLICATIONS FROM ONE PACKAGE.
	APL*PLUS II Developer System	3200	Unlimited distribution of APL*PLUS II Run-Time applications!
	APL*PLUS II for UNIX	poa	STSC's 2nd generation APL for all major Sparc and Risc Unix workstations.
	APL*PLUS VMS	poa	2nd generation APL for DEC VAX computers running under VMS.
	APL*PLUS Mainframe	poa	Enhances VS APL with many high performance, high productivity features. For VM/CMS and MVSTSO offers simple upgrade from VS APL.
Dyadic	Dyalog APL for DOS/386	995	Second generation APL for DOS.Runs in 32-bit mode, supports very large workspaces. Unique "window-based" APL Development Environment and Screen Manager. Requires 386/486 based PC or PS/2, at least 2Mb RAM, EGA or VGA, DOS 3.3 or later.
	Dyalog APL/W for Windows	995	As above, plus object-based GUI development tools. Requires Windows 3.0 or later.
	Dyalog APL for Unix	995-12,000	Second generation APL for Unix systems. Available for Altos, Apollo, Bull, Dec, HP, IBM 6150, IBM RS/6000, Masscomp, Pyramid, NCR, Sun and Unisys machines, and for PCs and PC/2s running Xenix or AIX. Oracle interface available for IBM, Sun and Xenix versions.
IAC/Human Interfaces-I-APL/Mac		13	Macintosh version of I-APL
I-APL Ltd	I-APL/PC or clones	8 - 11	ISO conforming interpreter. Supplied only with manual (see 'Other Products' for accompanying books).
	I-APL/BBC Master	8	As above
	I-APL/Archimedes	11	As above
	I-APL/Macintosh	13	As above
	Iverson Software Products		I-APL is the UK agent for ALL Iverson Software products, including: Sharp APL + enhancements to the ISI-APL standard Versions of J (and the associated texts) for PC, Mac, Arc, Atari. Please call for price list and order form. Please note there is a £3 packing charge on all orders.
IBM (APL Products)	APL Version 2 Release 1	poa	Full APL2 System for IBM 370 and 390. Product No. 5688-228. See your IBM Branch Office.
	APL2 Application Env't Ver2 Rel1	poa	Run-time Environment for APL2 Packages (IBM 370 and 390). Product No. 5688-229. See your IBM Branch Office
	APL2 for the RISC System/6000	poa	Product No. 5765-012. See your IBM Branch Office.
	APL2 for the IBM PC (US version)	\$495	Product No. 5799-PGG, Part No. 6242936, PRPQ No. RJB411 (in HONE, RJ0411). (from IBM Direct, order by Part Number; from your IBM Branch Office, order by PRPQ Number)
	APL2 for IBM PC (European vers)	poa	Product No. 5604-280, Part No. 38F1753
	TryAPL2	free	Free APL2 for educational/demonstration use. Write to APL Products, specifying diskette size desired.
IBM UK	IBM PC APL2	348	APL2 for the IBM PC. From all IBM dealers, including MicroAPL

Iverson Software Inc.	APL386	\$30	Sharp APL Release 20 for PC 386, 486 with graphics, and ability to operate under Windows.
	APLIPC	\$30	For PC
MicroAPL	APL.68000 Level I	2000	First generation APL with numerous enhancements. Multi-user version (Unix, Mirage, MCS).
	APL.68000 Level II	2500	Second generation APL. Nested arrays, user defined operators, selective specification etc. Multi-user version (Unix, Mirage, MCS)
	APL.68000/X	1500-6000	Second-generation APL. Nested arrays, user defined operators, selective specification, etc. Multi-user AIX version with full OSF/Motif support.
	APL.68000 Level I Mac, ST, Amiga	87	First generation APL. Single user, full windowing interface, software floating point support.
	Mac, Amiga	260	First generation APL. Single user, full windowing interface, hardware floating point.
	APL.68000 Level II ST	170	Second generation APL. Full windowing interface, software floating point support.
	Amiga	260	Second generation APL. Full windowing interface. Hardware and software floating point support.
	Mac	520	Second generation APL. Full windowing interface. Hardware and software floating point support.
	APL*PLUS Rel 10	450	
	APL*PLUS II V 4.0	1395	
Optima	APL*PLUS/PC	369	
	APL*PLUS II	950	
	APL*PLUS II PC Developers Kit	poa	
	Dyalog APL	999	
Reuters Ltd	SHARP APL for MVS	poa	for IBM MVS mainframes
	SHARP APL for Unix	poa	for IBM RS/6000 and Sun SPARC
Unitware	APL*PLUS/PC	495	STSC's full feature APL for IBM PC/XT/AT, Compaq, Olivetti.
	Run-Time	call	Closed version of APL*PLUS/PC which prevents user exposure to APL.
	APL*PLUS/UNIX	call	STSC's full feature APL for UNIX based computers
	APL*PLUS II	call	STSC's full feature APL for 386 machines.

## APL VISUAL DISPLAY UNITS AND PRINTERS

COMPANY	PRODUCT	PRICES (£)	DETAILS
APL People	IBM & compatibles	poa	IBM and 'budget' APL VDUs - monochrome/colour/graphics.
APL People	Epson series	200	Inexpensive dot-matrix and NLQ printers
	Quen-data & Qume etc	500	Daisy-wheel printers
Dyadic	IBM 3151	599	Monochrome APL/ASCII vdu with APL keyboard. Supports downloaded Dyalog APL font.
	IBM 6154	1,228	Colour APL/ASCII vdu with APL keyboard. Supports downloaded Dyalog APL font.
Dyadic	Printers (Various)	poa	Range of APL printers available.
General Software	Mellordata	poa	
Shandell	HDS Monitors	965-1395	A range of APL monitors. Please call for details.
	X-Terminals	1000-3500	View-stations for use with all X-Windows systems



## APL PACKAGES

COMPANY	PRODUCT	PRICES(£)	DETAILS
Active Workspace Ltd	Syndicate Manager	poa	Lloyd's managing agent's syndicate / company accounting system. Stamp & Personal accounts (inc. Run offs)
APL-385	APL-385 FSM-385 DRAW-385 DB-385 GEN-385	50(PC),125(mf)	Including ... Screen development Screen design Relational W.S. Miscellaneous Utilities
The APL Group	Qualedl	\$1500-4000	Electronic Data Interchange (EDI) translation software for the PC, with strict compliance checking.
APL Software Ltd (mainframe)	RDS	poa	Relation Data Base System
	IPLS	poa	Project Management System
	REGGPAK	poa	Regression Analysis Package
(microcomputer)	POWERTOOLS	295	Assembler written replacement function for commonly used CPU-consuming APL functions, includes a Forms Processor.
	REGGPAK	poa	Regression Analysis Package
	RDS	990	Relational Database System
Cocking/Drury (for VSAPL)	E'MENTS & SHAREFILE	poa	Component files, quad-functions & nested arrays for VSAPL under VM/CMS & MVS/TSO
	COMPILER	poa	The First APL compiler!
	FILEPRINT	poa	Print APL component files
	FILECONVERT	poa	Converts non-APL files to APL
	FILEMANAGER	poa	Extends APL primitives to database management
	TOOLS + UTILITIES	poa	APL Software development tools
	DATAPORT	poa	Information Centre spreadsheet incorporating data exchange between APL, FOCUS, IFPS, SAS, APL/DI, ADRSII, Lotus123, Visicalc, Multiplan & DIF
(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
	FMT	poa	Full featured FMT for APL2
	WSDOC	poa	Workspace documentation utilities
	FILEMANAGER	poa	Extends APL primitives to database management
(for PC's)	APL*PLUS PC Tools	275	Utilities including: RAM disk, full screen data entry, menu input, report generation, exception handling and games.
	IRMA Module	90	327x IRMA support.
	FIN & STAT. LIBRARY	250	Financial & Statistical routines
	SPREADSHEET MGR	150	APL-based spreadsheet for APL*PLUS/PC. Cell arithmetic; transfers to ASCII & Lotus
CYBEX AB	APL Graf/PC	290	Presentation graphics for APL*PLUS/PC (CGI)
	APL Graf II/PC	390	Presentation graphics for APL*PLUS II/PC (CGI).
	Utility Functions APL2	1900	For APL mainframe; incl. a very fast search.
	Utility Functions II/PC	130	Same package for APL*PLUS II/PC.
H.M.W.	4XTRA	poa	Front-end Foreign Exchange dealing / pos keeping
	Arbitrage	poa	Arbitrage modelling
	Basket	poa	Basket currency modelling

	Menu-Bar	poa	pull-down menu for APL*PLUS/PC
HRH Systems	APL Utilities	poa	Software to transfer workspaces between APL*PLUS and Sharp, and between APL*PLUS and I-APL. Software to import IBM .ATF files to APL*PLUS.
	APL*PLUS Utilities		Public domain software, unlock locked fns, a user-friendly alternative to locking, fns of mathematical physics, menus, and others.
IAC/Human Interfaces	IAC/Graf	15	Graph plotting for I-APL/Mac
	IAC/Vox	15	Spoken APL characters for I-APL/Mac
I-APL Ltd	Educational workspaces	5	PC format disks with the examples from: Thomson, Espinasse (Kits 1-4), Kromberg, Jizba & FinnAPL. All the examples to save your fingers!
Impetus Ltd	<i>Impetus</i>	poa	Corporate Modelling and Reporting System.
INFOSTROY	APL*PLUS/Xbase Interface (II/386 Version 2)	\$198	Complete package written in C. Comparable with the data, Index & memo files of FoxPro, dBASE, & Clipper. Multi-user support. No DBMS license required.
	(PC Version 2)	\$98	As above for APL*PLUS/PC.
	(DLL Version 1)	\$198	The same in a DLL form! Gives your Windows applications all advantages of DLLs.
Interprocess Systems	IEDIT	\$3000-5000	Full screen APL2 editor with immediate APL execution, and full-screen debugger
	AOC	poa	APL2 Optimizing Compiler, translates APL2 functions to FORTRAN programs.
(mainframe)	AFM	\$6500-15300	High performance component and keyed file system (VS APL and APL2)
(PC)	AFM	\$175	Single user component and keyed files for APL2/PC.
	Enhanced Format	\$2575	A QuadFMT data formatter for VS APL and APL2
	PowerCode	\$2000	External functions for APL2
	CALL/JAP	\$4700	For calling non-APL programs (VS APL and APL2)
	WSORG	poa	Full-screen Workspace Organizer for APL2.
Merica	STATGRAPHICS 5	635	The old favourite.
	STATGRAPHICS PLUS	895	Now running under APL*PLUS II - so you can have as much workspace as you want!
	LOGOL 90	poa	Logistics management system for 386/486 & RISC computers. Sales Forecasting, Inventory Management, Master Scheduling, Distribution Requirements Planning, Sales & Operations Planning.
	TWIGS	poa	A modular library of tools to teach and explore state-of-the-art materials management concepts. Developed by R.G. Brown.
MicroAPL	MicroTASK	250	Product development aids
	MicroFILE	250	File utilities and database
	MicroPLOT	250	Graphics for HP plotters etc
	MicroLINK	250	General device communications
	MicroFORM	250	Full screen forms design
	MicroSPAN	250	Comprehensive APL tutor
	MicroPLOT/PC	250	For APL*PLUS/PC product
	MicroSPAN/PC	250	APL self instruction for APL*PLUS/PC
	STATGRAPHICS Rel 5	590	
Reuters Ltd	GLCS	poa	Global risk management for banks
	LOGOS	poa	Application Development Environment
	MAILBOX	poa	Electronic Mail

	NEWSFLASH	poa	Real-time message exchange
	VIEWPOINT	poa	4GL with interfaces to DB2, ADABAS, and MVS datasets
UNIWARE (for mainframe)	STSC's ENHANCEMENTS	poa	Quad-functions & nested arrays for IBM VSAPL under VM/CMS and MVS/TSO
	STSC's SHAREFILE	poa	component files for IBM VSAPL under VM/CMS and MVS/TSO and for IBM APL2
	TOOLS & UTILITIES	poa	Including FILEPRINT, FILESORT, FILECONVERT FILEMANAGER(EMMA) STSC's database package
	EXECUCALC	poa	Mainframe spreadsheet compatible with VISICALC and part of LOTUS 1-2-3 under VSAPL(VM or TSO)
(for APL*PLUS/PC)	APL Debugger 2.1	FF1950 FF9750	A visual APL debugger to help develop applications (site license)
	Menus 3.0	FF2450 FF12250	Complete set of hierarchical menu utilities (site license)
	ETATGEN 2.0	FF1950 FF9750	Page layout report generator (site license)
	UNITAB 2.0	FF4550 FF22750	An APL*PLUS spreadsheet-like data entry and validation system (site license)
	UNIASM 3.0 (site license)	FF4950	Assembler utilities to speed up APL*PLUS/PC applications
	UNISTAT 5.1	FF2900	Data analysis add-on module for Statgraphics
(for APL*PLUS II)	UNIWARE Toolkit II 4.1	FF39000	(site license only). Relational database system and complete set of utilities for APL*PLUS II development
	APL Debugger II 2.1	FF2950 FF14750	A visual APL debugger to help develop applications (site license)
	Menus II 4.0	FF3950 FF19750	Complete set of hierarchical mouse-driven menu utilities (site license)
	ETATGEN II 2.0	FF2950 FF14750	Page layout report generator (site license)
	UNITAB II 2.0	FF6950 FF34750	An APL*PLUS spreadsheet-like data entry and validation system (site license)
	UNISTAT Plus 5.2	FF4300	Data analysis add-on module for Statgraphics
Warwick University	BATS	250	Menu driven system for time series analysis and forecasting using Bayesian Dynamic modelling. Price is reduced to £35 for academic institutions.
	FAB	free	Training program for the above.
Zark	APL Tutor (PC)	\$299	APL computer-based training. Available for APL*PLUS PC & APL*PLUS II. Demo disk \$10.
	APL Tutor (MF)	\$5000	Mainframe version.
	Zark ACE	\$99	APL continuing education. APL tutor news and hotline phone support.
	APL Advanced Techniques....	\$59.95	488pp. book, (ISBN 0-9619067-07) including 2-disk set of utility functions (APL*PLUS PC format).
	Communications	\$200 pc, \$500 mf	Move workspaces or files between APL environments.

**APL CONSULTANCY**

COMPANY	PRODUCT	PRICES(£)	DETAILS
Active Workspace	Consultancy	poa	PC Based APL system design, programming and implementation.
Adfee	Consultancy	poa	Development, maintenance, conversion, migration, documentation, of APL products in all APL environments

APL People	Consultancy	poa	Consultants available at all levels, with experience in: VS APL, APL*PLUS, APL2, Sharp APL, Dyalog APL, APL6800, C/Unix, TSO/MVS, VM/CMS, graphics, Operational Research etc. Expertise in APL system design, project management, prototyping, financial applications, decision support systems, MIS, links to non-APL systems, documentation, etc.
Camacho	Consultancy	poa	Specialising in programming & manual writing.
Chapman	Consultancy	250-500	24-hour programmer: APL, C, assembler, graphics; PC, mini, mainframe, network.
Cocking/Drury	Consultancy	175-275 275-350 300-450 400-600 450-750	Junior consultant Consultant Senior consultant Principal Consultant Managing consultant
Peter Cyrilax	Consultancy	100-150 120-200 160-300	Junior Consultant Consultant Senior Consultant
David Crossley	Consultancy	poa	Broad experience in many APL environments
Dyadic	Consultancy	poa	APL and Unix system design, consultancy, programming and training.
E & S	Consultancy	poa	System prototyping: all types of information system, engineering software, graphics and decision support systems APL*PLUS/PC, APL2, Dyalog APL
General Software	Consultancy	from 120	
Michael Hughes	Consultancy	poa	Consultant with 10+ years experience with various APL interpreters and C.
H.M.W.	Consultancy	poa	System design consultancy, programming. HMW specialize in banking and prototyping work.
IAC/Human Interfaces	Consultancy	350	APL on Macintosh & PC. HCI design. VDU ergonomics: EC/Health & Safety compliance.
	Documentation	100-200	On-line assistance, product demos & mock-ups, manual writing; foreign language software localization.
	Training	poa	Using I-APL for courseware & distance learning materials; Mac programming in C, APL & HyperCard.
INFOSTROY	Consultancy	poa	APL*PLUS & Windows consultancy. Porting of software written in C into APL*PLUS.
Intelligent Programs	Consultancy	poa	Systems development, enhancements, support.
	Documentation	poa	Preparation of new manuals, rewriting of existing materials.
	Training	poa	Training for APL experts through to non-technical system users.
Kestrel	Consultancy	poa	All APLs, all environments. Design, analysis, coding, maintenance, documentation, training, interfacing.
Lingo Allegro USA	Consultancy	poa	General APL consultancy specializing in Prototyping, Migration, Mainframe to PC Downsizing, Performance Analysis, Troubleshooting, and Graphics.
MicroAPL	Consultancy	poa	Technical & applications consultancy.
Ellis Morgan	Consultancy	250-500	Business Forecasting & APL Systems.
M.T.I.C.	Consultancy	240-500	Business analysis and APL consultancy
Optima	Consultancy	poa	A range of consultants with 3-15 yrs APL PC and mf experience.
Parallax Systems Inc	Consultancy	\$750	Introductory APL, APL for End-user & Advanced Topics in APL
QB On-Line	Consultancy	350	Specialising in Banking, Financial & Planning Systems.
Rochester Group	Consultancy	poa	Specialise in MIS using Sharp APL
Rex Swain	Consultancy	poa	Independent consultant, 15 years experience. Custom software development & training, PC and/or mainframe.
Uniware	Consultancy (Senior)	FF/day 5000	Consultancy from people with at least 8 years APL experience.
	Consultancy (Senior)	FF/day 7500	Advice and training in Windows programming with APL*PLUS II

	Training	FF10000	5-day class on Windows programming with PLUS II version 4.0
Wickliffe Computer	Consultancy	poa	System design, consultancy, programming and documentation. Especially project management and decision support systems

## OTHER PRODUCTS

COMPANY	PRODUCT	PRICES(£)	DETAILS
Adfee	Employment	poa	Contractors and permanent employees
APL People	Employment Agency	poa	Permanent employees placed at all levels. Contractors supplied for short/long-term contracts, supervised or unsupervised. Executive Search service available.
HMW	Employment	poa	Contractors and permanent employees placed.
HRH Systems	APL lessons		On-screen Interactive APL lessons for APL*PLUS, TryAPL2, Sharp and I-APL — in English or French.
	The BBS/APL:		301-384-3672, 300/1200/2400/9600 b, N-8-1, 24 hours. APL educational material is downloadable free. An additional 30 megs of APL software for APL*PLUS, PLUS II, IBM, Sharp and I-APL is available to subscribers (cost is \$24/yr).
I-APL Ltd	An APL Tutorial	3	45pp by Alvord & Thomson
	An Encyclopaedia of APL (2d Ed)	6	228pp by Helzer
	APL In Social Studies	3	36pp by Traberma
	I-APL Instruction Manual (2d Ed)	3	55pp by Camacho & Ziemann
	APL Programs for the Mathematics Classroom (Springer-Verlag)		
		16	185pp by Thomson
	J Dictionary	16	by Ken Iverson
	Programming in J	10	75pp by Ken Iverson
	Arithmetic	12	118pp by Ken Iverson
	An Introduction to J	8	47pp by Ken Iverson
	Tangible math	8	36pp by Ken Iverson
	Sharp APL Reference Manual	18	349pp by Berry
	APL Press Books	poa	A comprehensive selection of early APL literature
			<i>Please note there is a packing charge of £3 per order</i>
IBM (APL Products)	APL2 Keypaps	poa	Keypaps for the PC and PS/2 (USA and UK standard). Product Number SX80-0270.
	APL2 Keyboard Stickers	poa	Product Number SC33-0604.
Iverson Software Inc. J		\$24	Documentation & shareware for many popular machines, including PC, MAC, SUN, & ARCHIMEDES.
	J Source	\$90	Documented source code in C for porting to other machines (Shareware).
	Programming In J	\$15	76pp.
	Tangible Math	\$12	34pp.
	Arithmetic	\$18	123pp
Kestrel	Employment	poa	Permanent and contract, home and abroad. From individual placement to supply of complete project teams.
	Software Library	poa	Low-cost software distribution service; call for details.
Renaissance Data Systems	Booksellers		The widest range of APL books available anywhere. See Vector advertisements.
Reuters Ltd	MVSLINK	poa	Auxiliary processor complex, providing SharpAPL (Unix & MVS) with facilities to access MVS data, and invoke non-APL MVS software on remote MVS systems.
	SSQL	poa	Auxiliary processor providing SharpAPL (Unix & MVS) with high-performance SQL interface to DB2.

**OVERSEAS ASSOCIATIONS**

GROUP	LOCATION	JOURNAL	OTHER SERVICES	Ann.Sub.
ACM/SIGAPL	USA	Quote Quad		
APL Bay Area	USA N. California	APLBUG	Monthly Meetings (2nd Monday)	\$15
APL Club Germany	Germany	APL Journal	Semi-annual meetings	DM60
Ass. Francophone pour la promotion d'APL	France	Les Nouvelles d'APL		
BACUS	Belgium	APL-CAM	Conferences & Seminars	£18 (\$30)
CPC UG APL SIG (Capital PCUG)	Washington, D.C.	Capital PC Monitor	Monthly meetings, occasional classes	free
Dutch APL Assoc.	Holland	-	Mini-congress, APL ShareWare Initiative	
APL Club Austria	Austria	-	Quarterly Meetings	200AS(indiv), 1000AS(corp)
FinnAPL	Finland	Newsletter		
SwedAPL	Sweden	SwedAPL Nytt	Semi-annual meetings, seminars	SEK 75
Sydney APLUG	Sydney, Australia	Epsilon	Monthly Meetings	

**VENDOR ADDRESSES**

COMPANY	CONTACT	ADDRESS & TELEPHONE No.
ACM/SIGAPL		ACM, 1515 Broadway, New York, NY 10036 USA Tel: (212) 626-0611
Active Workspace Ltd	Ross D Hanson	Moulsham Mill Centre, Parkway, Chelmsford, Essex, CM2 7PX. Tel: (0245)-496647; Fax: 0245-496646.
Adfee	Bernard Smoor	Dorpsstraat 50, 4128 BZ Lexmond, Netherlands. Tel: 31-3474-2337, Fax: 31-3474-2342
APL-385	Adrian Smith	Brook House, Gilling East, York. Tel: 04393-385
APLBUG	Lewis H. Robinson	1100 Gough St, Apt 14A, San Francisco, CA 94109, USA Tel: (415) 928-2058
APL Club Austria	Erich Gall	IBM Österreich, Obere Donaustrasse 95, A-1020 Wien, Austria
APL Club Germany	Dieter Lattermann	IBM Germany, Wilckensstrasse 1a, D-6900 Heidelberg, Germany. Tel: (49) 6221-404-243
The APL Group Inc	Stuart Sawabini	644 Danbury Road, WILTON, CT 06897 USA. Tel: (203) 762-3933 Fax: (203) 762-2108
APL People / Software	Jill Moss	The Old Malthouse, Clarence St, BATH, BA1 5NS. Tel: 0225-462602
Association Francophone pour la promotion d'APL	Dr. Gérard Langlet	SCM, C.E. Saclay, F-91191-Gif sur Yvette, France. Fax: (33) 1 69 08 79 63
BACUS	Joseph de Kerf	Rocinberg 72, B-2570 Duffel, Belgium.
Anthony Camacho		11 Auburn Road, Redland, Bristol BS6 6LS. Tel: 0272-730036.
Paul Chapman		41 Lams Conduit Street, London WC1N 3NG. Tel: 071-831-3762.-
Cocking & Drury Ltd.	Romilly Cocking	180 Tottenham Court Road, LONDON, W1P 9LE Tel: 071436 9481 Fax: 071-436 0524
CPC UG	Lynne Startz	Capital PC User Group, 51 Monroe Street, Suite PE-2, Rockville, Maryland 20850, USA. Tel: 301-762-9372.
CYBEX AB	Lars Wentzel	Gruvgatan 35B, S-421 30 V. Frölunda, Sweden. Tel: (46) 31-45 37 40. Fax: (46) 31-45 24 23.
Datatrade Ltd.	Ian Tomlin	1 & 2 Sterling Business Park, Salthouse Road, Brackmills, Northampton, NN4 0EX. Tel: 0604-760241
David Crossley		187 Le Tour du Pont, Quartier Le Mourre, 84210 ST DIDIER, France Tel: 90-66-08-87
Dutch APL Association	Bernard Smoor (Sec)	Postbus 1341, 3430BH Nieuwegein. Tel: 03474-2337
Dyadic Systems Ltd.	Peter Donnelly	Riverside View, Basing Road, Old Basing, Basingstoke, Hants RG24 0AL. Tel: 0256-811125 Fax: 0256-811130
E & S Associates	Frank Evans	19 Homesdale Road, Orpington, Kent BR5 1JS. Tel: 0689-824741
FinnAPL	Jantunen Veli-Matti (Sec)	Myllytie 10 A 7, 04400 Järvenpää, Finland Tel: 90-2916470 (home)

General Software Ltd	M.E. Martin	22 Russell Road, Northholt, Middx, UB5 4QS. Tel: 081-864-9537
H.M.W. Trading Systems	Stan Wilkinson	Hamilton House, 1 Temple Avenue, Victoria Embankment, London EC4Y 0HA. Tel: 071-353-4212; Fax: 071-353-3325
HRH Systems	Dick Holt	3802 N Richmond St, Arlington, VA 22207, USA. Tel: (703) 528 7624 BBS/APL Tel: (703) 528 7617 1200-14400b, N-8-1, v.32 24hr (v.32bis) Internet: dclck.holt@acm.org
Michael Hughes		28 Rushton Road, Wilbarston, Market Harborough, Leics., LE16 8QL Tel: 0536-770998
IAC/Human Interfaces	Ian A. Clark	9 Hill End, Frosterley, Bishop Auckland, Co. Durham DL13 2SX Tel: 0388-527190. Email: clark.i@applelnk.apple.com
I-APL Ltd	Anthony Camacho (for queries, order forms)	11 Auburn Road, Redland, Bristol BS6 6LS. Tel: 0272-760036
	J C Business Services (for pre-paid orders only)	56 The Crescent, Milton, Weston-super-Mare, Avon, BS22 8DU
IBM (APL Products)	Nancy Wheeler	APL Products (M46V/D12), IBM Santa Teresa, PO Box 49023, 555 Bailey Avenue, San Jose CA 95161-9023, USA. Tel: 1-408-463-APL2
Impetus Ltd	Cedric Heddle	Rusper, Sandy Lane, Ivy Hatch, SEVENOAKS, Kent TN15 0PD Tel: 0732-885126
INFOSTROY	Alexei Miroshnikov	3 S. Tulenin Lane, St. Petersburg 191186 Russia. Tel:+7 812-3111611 Fax:+7 812-3153321 Email:aim@infostroy.spb.su
Interprocess Systems Inc.	Stella Chamberlain	11660 Alpharetta Highway, Suite 455, Roswell, Georgia 30076, USA Tel: (404) 410-1700. Fax: (404) 410-1773
Intelligent Programs Ltd	Mike Bucknall	9 Gun Wharf, 130 Wapping High St, London E1 9NH Tel: 071-265-1120
Iverson Software Inc.	Eric Iverson	33 Major Street, Toronto, Ontario, Canada M5S 2K9 Tel:(416) 925-6096 Fax: (416) 488-7559
Kestrel Consulting	Mark Harris	Business & Technology Centre, Bessmer Drive, Stevenage, Herts SG1 2DX Tel:0438-310155 Fax:0438-310131 E:kestrel@apl.demon.co.uk
Lingo Allegro USA Inc.	Steven J. Halasz	203 North LaSalle St., Suite 2100, Chicago IL 60601 USA Tel: (312) 558-1342 Fax: (312) 346-9603
Mercia Software Ltd.	Gareth Brentnall	Holt Court North, Heneage Street West, Aston Science Park, Birmingham B7 4AX. Tel: 021-359-5096. Fax: 021-359-0375
MicroAPL Ltd.	David Eastwood	South Bank Technopark, 90 London Road, LONDON SE1 6LN Tel: 071-922 8866 Fax: 071-928 1005
Ellis Morgan		Myrtle Farm, Winchester Road, Stroud, Petersfield, Hants. Tel: 0730-263843
M.T.I.C.	Ray Cannon	21 Woodbridge Rd, Blackwater, Camberley, Surrey GU17 0BS Tel: 0252-874697
Optima Systems Ltd	Paul Grosvenor	Airport House, Purley Way, Croydon, Surrey CR0 0XY Tel: 081 781-1812 Fax: 081 781-1999
Peter Cyriax Systems	Peter Cyriax	22 Hereford Road, London W2 4AA. Tel: 071-229-5344
QB On-Line Systems	Phillip Bulmer	5 Surrey House, Portsmouth Rd., Camberley, Surrey, GU15 1LB. Tel: 0276-20789. Fax: 0276-683427. Mobile: 0831-307548
Renaissance Data Systems	Ed Shaw	P.O. Box 20023, Park West Finance Station, New York, NY 10025-1510, U.S.A. Tel: (212)-864-3078
Reuters Ltd	Laurie Howard	Reuter Nederland BV, PO Box 8230, 1005 AE Amsterdam, Netherlands Tel: +31 20 570 8733 Fax: +31 20 570 8758
The Rochester Group Inc.	Robert Pullman	50 S.Union St., Rochester NY 14607-1828, U.S.A. Tel: 716-454-4360. Fax:716-454-5430
Shandell Systems Ltd.	Maurice Shanahan	Chiltern House, High Street, Chalfont St. Giles, Bucks., HP8 4QH. Tel: 02407-2027. Fax: 02407-3118
Rex Swain		8 South Street, Washington, CT 06793, U.S.A. Tel: 203-868-0131
SwedAPL	Glan Medri	Box 16181, S-103 24 Stockholm, Sweden Tel:+46 (8) 96 09 47
Sydney APLUG	Rob Hodgkinson	GPO Box 1425, Sydney, NSW 2001, Australia
Uniware	Eric Lescasse	15 Rue Erlanger, 75016 Paris, France. Tel: (1) 45-27-20-61. Fax: (1) 45-27-20-71. Telex: 648348F UNIWARE
Wickliffe Computer Ltd	Nick Telfer	76 Victoria Rd., Whitehaven, Cumbria, CA28 6JD. Tel: 0946-692588
Warwick University	Prof. Jeff Harrison	Dept of Statistics, University of Warwick, Coventry, CV4 7AL Tel: 0203-523369
Zark Incorporated	Gary A. Bergquist	53 Shenlpsit St, Vernon CT 06066, USA. Tel: (203) 872-7806

# We just couldn't leave well enough alone!

APL\*PLUS II Version 5.1

Announcing: APL\*PLUS II  
Version 5.1

with 3-d Controls!

Graphics!

Icons!

Toolbars!

Plus:

- Windows Common Dialog Boxes
- Instant Start-up Windows!
- APL TrueType Font
- Fast Form Design

Bitmaps!

**APL\*PLUS® II for DOS and Windows™** just got better! It's now even easier (and a lot faster!) to build applications using *all* of the features of Windows 3.1. There's even a Software Developer's Kit that shows you how to *easily* add third-party custom controls!

And don't forget that **APL\*PLUS II** is *the only* APL system that provides you with a fully-integrated Debugger, Paradox Interface, full DDE access, Lotus

and dBase Import/Export, optimized Assembler functions, the User Command Processor, a Numeric Data Editor and the most powerful session manager available anywhere!

**Get GUIng today!** In the UK, phone Cocking & Drury at 071-436-9481. Elsewhere, call your local dealer or contact Manugistics, Inc. in the U.S. at (301) 984-5412, Fax (301) 984-5094 for the name of a dealer near you.



# APL\*Plus II/386 Version 5.0

*reviewed by Dave Piper*

## Introduction

Version 1 of APL\*Plus II/386 was the subject of two reviews in Vector 5.4 [Cyriax 5.4, Askoolum 5.4]. Version 3 was reviewed in Vector 8.1 [Pearson 8.1], version 4 in 9.1 [Crossley 9.1]. All being well, this review will appear in 9.4; we should be due for version 6 sometime around Vector 10.3 — next Christmas! Rumours of version 5.1, due around March/April, already abound.

The evolution (no pun intended) of the product over these releases stands some examination. Listed below are edited highlights of the new features in the most recent versions:

- |             |  |
|-------------|--|
| Version 3   | Session manager and interpreter unified<br>□WA enhancements<br>"tools" as part of the product<br>mouse support<br>user command processor                             |
| Version 3.5 | Windows 3.0 tolerance  |
| Version 4.0 | Windows interface<br>Paradox interface<br>character based "sub-windows"<br>evolution towards APL2<br>bound documentation   |
| Version 4.1 | OS/2 tolerance (dongle!)   |
| Version 5.0 | Windows 3.1 windowed character set support<br>further evolution<br>enhanced windows interface<br>APLGUI<br>interactive debugger<br>automated installation<br>GRAFLIB |

A slight disclaimer — this review is based on my experience of version 5.0 so far. Having only had the product for a few weeks, a fully comprehensive examination has not been possible. In particular, I have had no opportunity to examine GRAFLIB or the related user commands.

## The Language

Not a lot has changed here. If you are coding at evolution level 2, then a few steps toward the APL2 IBMard [Piper 9.1] have been taken. Specifically, `enclose` and `disclose` now permit axis specification. One useful side effect of this is that (at long last [sorry Jerry]), we have length tolerance when disclosing a nested object into a higher rank array. One of the bug-bears of *MIX* (and its successor `⊖MIX`) is that all items being mixed must have the same length.

Interestingly, `⊖MIX` still retains its original behaviour — it appears that `disclose` and `mix` have now parted company within the interpreter. In version 4.0, `⊖MIX` and `DISCLOSE` were aliases of each other, providing the same functionality.

APL\*Plus II/386 does not follow either APL2 or APL2/PC when generating fill items for mixed arrays. The following example illustrates the 3 different rules:

```
DISPLAY 3 3+2 2p1 'A' 'B' 2
```

APL\*Plus II/386

1	A	0
B	2	0
0	0	0

APL2/PC

1	A	0
B	2	
0	0	0

APL2

1	A	0
B	2	
0		0

APL\*Plus II/386 always uses the first item of the array to generate fill items; APL2/PC uses the first item of a row if it already exists, otherwise the first item of the array; APL2 uses the first item of a row if it already exists, or the first item of the column. [C&D 1].

One unfortunate syntactic exception is still present at evolution level 2; pure numeric strands can be indexed without parentheses:

```
1 2 3[2]
2
```

Any other type of strand must be parenthesized before indexing can occur.

Both language bugs that I have found in previous versions still exist. In evolution level 1, indexing into strands can go wrong:

```

[]J0+1
c+3
1 2 c ≡ 1 2 3           A They match
1
1 2 3[1]                 A Index first item
1
1 2 c[1]                 A Likewise ?
1 2
1 2 c[2]                 A Second item !
3
1 2 c[3]                 A Third item !*?
INDEX ERROR
1 2 c[3]
  ^   ^

```

This cannot happen at evolution level 2 since non-numeric strands must be parenthesized before being indexed.

The second problem involves the strand assignment of non-simple scalars:

```

(a b c)+3ρc''           A Each gets an empty
≡''a b c                 A Check the depth of each
1 1 1
ρ''ρ''a b c              A Check the rank of each
1 1 1
(a b c)+c''             A Scalar extension we hope!
≡''a b c                 A Check the depth of each
2 2 2
ρ''ρ''a b c              A Check the rank of each
0 0 0

```

When we rely on scalar extension, each name assigned gets the scalar rather than the content of the scalar.

### Windows 3.1 Support

Version 5.0 offers full support for operation as a DOS application under Windows 3.1. The support provided by Windows for DOS applications was considerably enhanced in release 3.1, so this is a very desirable move forward. When running in a window (rather than full screen mode), the mouse performs the same functions as if the program were hosted in DOS or running in full screen mode under Windows. This contrasts with the rather useless "mark/edit" mode adopted in Windows 3.0. To make use of the new mouse functionality, mouse driver 8.2a or later is required.

Version 5.0 introduces support for the APL character set when running in a window under Windows 3.1. Character set support for Windows 3.0 was already

available in version 4.0, these fonts were not compatible with Windows 3.1. Windows 3.0 character set support is still available. Note, as a result of the way Windows supports fonts in DOS applications, all DOS applications will use the APL font if it is installed. This may result in some strange effects in the displays generated by other DOS applications. Most of the characters are quite well mapped, so really glaring changes should be kept to a minimum.

The 3.1 character set is available in several font sizes, ranging from a very thin but still usable 8×8 pixels to a large and well formed 16×8 pixels. Other, still larger, sizes are available but the full DOS display is not visible and must be scrolled using the scroll bars generated automatically by Windows.

The 8×8 font is quite difficult to read for long periods of time, but is good for working in a window which is less than half of the full display size. Using this font makes it possible to have other windows such as file manager opened to a significant extent and visible at the same time as the APL session.

## Printing

Direct printing using Windows facilities and a Windows printer font is still not supported in version 5. With the advent of a competent Windows interface (see below), the provision of a real Windows printer font would have been a great advantage. Fortunately, thanks to the sterling work of Adrian Smith and Ian Clark, TrueType APL fonts are now available. This article was produced using Word for Windows, using Adrian's font for the APL bits.

Support for DOS-based printing has been enhanced somewhat. Rather than down-loading the printer character set using the *PRINTERS* workspace (and associated component file), specific .COM files are available for a range of supported printers. Each of these downloads the character set and initialises the "port 10" printing facility. These programs can be re-executed at any time to reinitialise the printer if it has been reset by another program or user on a network.

## Installation

A significant new feature of version 5 is the provision of an installation program. This is a DOS program which allows the user to selectively install parts or all of the system. Different directories can be nominated for the various "parts" of the APL system. The structure of the directories is fixed — that is the files which constitute a "part" of the system cannot be varied (though subsets such as specific device drivers can be nominated). For those of us who have been users

for some time and wish to retain our existing directory structures it's back to the manual method.

Given that a file containing a list of all files in the system is provided it is a shame that this is not used to drive the installation process. Customisation of the file would allow any directory structure to be created. Better still, if the install program had been available from version 1...

## Enhanced Windows Interface

The updated Windows interface has overcome many of the problems discussed by Ray Cannon [Cannon 9.3]. In particular, a considerable amount of work has been put in to improve the speed of the interface. This release implements the agent using a virtual device driver (VAPLD.386) rather than relying on the Windows "timer" facility. The rate of call processing has gone up by at least a factor of 10; it is now possible to write programs with a Windows front end that respond as quickly as those written in compiled code. Only if filters are set up to process very frequent messages (mouse move or key depression) is any sluggishness really noticeable.

Various other enhancements have been made such as the use of a compiled .INI file containing all the Windows function, constant and message definitions (well nearly all!). Essentially, though, this remains an interface for real programmers; you must be prepared to get very intimate with the insides of Windows. Having climbed this not inconsiderable mole-hill myself in the last 10 weeks or so, I can attest to it being a frustrating and occasionally painful process. Of course, if you already know Windows, this interface could present you with many benefits in terms of flexibility and performance over the higher level interfaces provided by Dyalog APL and the APLGUI facility.

## APLGUI

APLGUI is new in Version 5. It provides a higher level, "object-oriented" interface for Windows programming. The interface uses one high level function to process all the methods and messages associated with a window object. Of course, APL\*Plus II has not suddenly turned into any object oriented system; all the mechanics of message passing and processing are implemented using real APL at a lower level.

The number of global variables and functions needed to support this framework is *astounding* — when using the GUI demo workspace, the total number of objects is *in excess of 1,000!* Managing such a large number of functions and

variables is no mean feat of itself — certainly solving obscure problems lying on the border between the supplied code and your own functions would be great fun. Perhaps of greater concern is the quality of the code used to implement the system. It seems to me, having studied significant volumes of code written by Manugistics, that they simply do not know how to write easily understood, maintainable code. Sorry chaps!

The basic object in the GUI system is the form — this corresponds to a dialog box in Windows parlance, and the “methods” (i.e. functions) required to make it work. A form contains controls of the normal types found in Windows — buttons, list boxes, edit fields etc. Rather confusingly, the standard control names are sometimes dropped in favour of GUI’s own names, so that radio buttons become “options”, static text becomes “label” and so on.

Messages (i.e. events [nearly]) can be sent to or received from either the main form or the child controls within it. Methods can be defined to process any of the supported messages. *Supported* is the key word here — if the message you wish to process is not supported then you have to find some other way around the problem, or start using low level Windows calls.

Having looked briefly at the GUI system, it is certain that low level Windows calls would have to be used, at some stage, to achieve all the effects required in any significant application. Certainly insufficient messages are supported for the more complex controls such as edit boxes. For example, there is no (GUI) way for a function to tell an edit field to cut the currently selected text.

Just to compound the effect of these difficulties, the interface is not overly robust. During early experimentation many hangs were generated. Even when we had gained expertise it was very easy to generate “ghost” windows which could not easily be disposed of. If your application collapses with an APL error, *)RESET* is not sufficient, the object oriented interface must also be reinitialised; this could make error handling in applications extremely complex and difficult to manage.

Objects, properties (attributes of the objects), messages and methods are linked together using a naming convention. A form (or a child control) has a caption property. We can reference or set the caption using the following statements:

```
caption←Win 'FmMine:caption'           A Reference
Win 'fmMine:caption' 'New Caption'    A Set
```

Similarly, for a child control:

```
child_cap+Win 'fmMine.bnOK:caption'  R Reference
Win 'fmMine.bnOK:caption' '~OK'      R Set
```

Methods follow a similar naming convention. For example, the SHOW method for a form would be:

```
fmMine_Show
```

The CLICKED method for a push button would be:

```
fmMine_bnOK_Clicked
```

The defined functions which implement methods do not take arguments, nor do they return explicit results. Instead, arguments are passed in the global variable *wArg* and the result set in the global *wZ*. This insistence on the use of global variables seems unfortunate from a code quality point of view; perhaps it would have been better to insist that all methods were implemented using monadic, result-returning functions.

APLGUI provides an interface at a level similar to that provided by Dyalog APL [Pearson 9.2], but currently rather less usable. The large number of objects in the workspace and the resulting object management difficulties (and problems with responsiveness) need to be solved before APLGUI can be regarded as a viable alternative to the low-level interface. Perhaps it is best to regard the current version of APLGUI as "beta-test", hopefully with better to follow.

### Interactive Debugger

Another new feature in Version 5. The debugger is integrated into the interpreter and can be toggled on and off from the session manager menus. Halt points and watch points are both supported; both remain set even when the debugger is inactive, but only take effect when it is used.

Halt and Watch points offer several advantages to the traditional APL trace/stop facility:

- can be disabled without being removed
- conditional operation
- halts apply to statements not function lines
- watch points can ignore execution context

When a halt or watch takes effect, the character based sub-windows introduced in Version 4 are used to display status information such as the:

- execution stack
- current halt/watch point

In addition, it is possible to display the values of the arguments at the current position. Options are offered on the status line permitting step-wise execution, execute to next statement, next line and so on.

How useful is it? Here I have to make a confession — I have only used it a couple of times in anger. In general, the standard trace and stop facilities provide me with sufficient control for solving any problems, especially since they are so easy to set within the ring editor. The only problems I suffer with trace and stop are the volume of output sometimes generated by trace and the number of unwanted stops that may occur in a looping or recursive function. The debugger provides exactly the right tools to avoid these problems. When I can get away with trace and stop I do, because they are so easy to set up and remove.

One annoying result of the introduction of the debugger is that the stop toggle has been moved on the keyboard. I think a quick dive into my config file to remap the keyboard is required.

## Bench Marks

The bench marks would have been copied from Adrian's review of APL\*Plus/PC Version 9 in Vector 6.4 [Smith 6.4]. Benchmarks have always been of questionable value, since the most valuable benchmark is often the perception of speed. Running under Windows makes performance measurement even more meaningless — some of the tests listed gave execution times varying by as much as 50% between the slowest and fastest times.

How does it feel? The answer must be just as fast as ever. One thing is for certain, the new primitives (especially *ENLIST* and *DISCLOSE*) offer significant performance benefits. Comparing our "make vector-of-character-vectors into matrix" with the primitive *DISCLOSE* indicates a potential ten-fold benefit — even after the defined function was carefully optimised!

## Conclusions

So is it all worth it? The broad answer must be yes; we now have a usable Windows interface that can create real Windows applications, and an integrated



debugger that provides many powerful facilities which are useful at least some of the time. There is a higher level interface to Windows with which we can experiment, even if it does not yet provide a real application development tool. It is possible to use the interpreter in a window, and see the APL characters, a further improvement in productivity for Windows users.

The remaining new features represent increments in usability rather than a productivity revolution. The debugger certainly does not provide a productivity revolution, not because of any lack of functionality but simply because debugging well written APL is usually quite easy. I suspect it will turn out to be of most use to those who are relatively inexperienced and tend to create poor quality code (perhaps Manugistics created it to support their own efforts?).

To sum up — it feels more like a minor release than a brand new version.

## References

- [Askoolum 5.4] APL\*Plus II for 80386 PCs.  
Vector 5.4, April 1989; pp 61-66
- [Cannon 9.3] APL\*Plus II/386 Interface to Windows.  
Vector 9.3, January 1993; pp 44-54
- [C&D 1] APL2 Nested Arrays Course  
Cocking & Drury Ltd 1990, 1991
- [Crossley 9.1] APL\*Plus II/386 Release 4  
Vector 9.1, July 1992; pp 53-60
- [Cyriax 5.4] APL\*Plus II — STSC's Second Generation APL System for 80386  
Machines.  
Vector 5.4, April 1989; pp 55-60
- [Pearson 8.1] APL\*Plus II Release 3  
Vector 8.1, July 1991; pp 73-76
- [Pearson 9.2] Dyalog APL/W  
Vector 9.2, October 1992; pp 55-64
- [Piper 9.1] General Correspondence  
Vector 9.1, July 1992; pp 9-12
- [Smith 6.4] APL\*Plus/PC Release 9  
Vector 6.4, April 1990; pp 54-57

# A First Look at the ISI APL/Windows (APLIWIN) Beta Release

by *Martin Gfeller*  
*mgf@rmx.risk.reuter.de*

## SHARP APL under Windows

When I.P. SHARP Associates ported their SHARP APL from the mainframe to the PC, they chose the unusual route of emulating an IBM/370 mainframe on the PC, and running the unchanged mainframe APL under it. This produced a solid, but very slow, implementation of APL. It was definitely too slow to run the large applications typical of the SHARP APL mainframe environment.

SHARP APL/PC never became commercially important in the APL arena, and was subsequently reclassified as shareware. Iverson Systems Inc. (ISI) acquired the rights to the product from Reuters, and developed several new releases, making it faster and better integrated into the PC architecture.

ISI APL/Windows (APLIWIN) is a major step in that effort, for it seamlessly integrates SHARP APL into Windows. At its core, it still emulates a /370 mainframe, but by using the 32-bit protected mode of the 386 architecture and Windows enhanced mode, it does this much quicker than its predecessors, and also provides large workspaces.

The following notes are based on the Beta release copy of October 1992, marked as APL release 5.0. ISI's Eric Iverson told me that a new release is planned for April, with significant improvements in the Windows interface and corrections of some of the deficiencies that are mentioned in this article.

## Language and System Features

APLIWIN features the same language elements as SHARP APL Version 20. This includes nested arrays, operators, packages, a component file system and complex numbers. However, it suffers from the same implementation restrictions as does the mainframe version: operators do not accept defined or derived function arguments and defined operators do not exist. These and some other restrictions have been removed in SAX, the Unix implementation of SHARP APL.

An aside about language differences: The SHARP APL language is different from the APL2 family in many respects. It emphasizes the uniform rectangular and multidimensional nature of arrays over arbitrary nesting. A very important concept of control is the rank of subarrays on which a function is applied directly; in APL2 on the other hand, the depth is more important. The rank concept assumes that a regular, rectangular structure is the normal case, whereas APL2's nesting and depth concept assumes nonuniformity as normal (see J.P. Benkard, *Nested arrays and operators*, APL'92 Conference Proceedings, St. Petersburg, July 1992, p.7).

Shared variables and a bundle of Auxiliary Processors were available in SHARP APL/PC under DOS, implementing a somewhat crude multitasking scheme, which was not easy to use from APL. This approach has been abandoned for Windows, and has been replaced by a handful of quad-functions. One family of thirteen system functions provides access to DOS files; some others translate between APL characters and ASCII, and provide a Window editor and link to the Window Driver (see below).

## Windows Features

The Session Manager and Window Driver are the same as provided with the PC version of J.

- **Session Manager and Editor:** A simple yet sufficient session manager provides for an input area and an editable and scrollable session transcript in a window. It allows cut and paste into the clipboard through DDE.

A self-contained cover function to invoke an editor is provided in a workspace. The editor handles both functions and text variables.

- **Window Driver:** A quad-function and a cover function interface to the Windows GUI through a little language of its own. The reference for this language takes eight pages in the documentation, and a description is beyond the scope of this *First Look*. Briefly, the approach taken resembles PostScript, in that a program is represented as human-readable ASCII text. It has some object-oriented features, as is to be expected for controlling a hierarchical windows environment. The language supports graphics, and a demo workspace gives some nice illustrative examples of these features.

Although there is no comprehensive workbench for building a user interface, the basic tools to build an application interface that takes full advantage of the Windows GUI are available.

- **TrueType APL font:** The APL font supplied with the product can be used from all Windows applications, and looks nice. Being a TrueType font, it scales well to any size.
- **Printing:** There is no built-in command to print. Text to be printed can be copied through the clipboard to an application such as *Windows Write*. Alternatively, a function supplied in a workspace writes an object to a file and invokes Write through the Window Driver. With the supplied TrueType APL font, good looking APL documents can be produced in a flexible way. However, the facility cannot be used for printing reports without user involvement.
- **Debugger:** Although SHARP APL includes the venerable trace and stop vectors, APLIWIN provides a much more powerful debugging aid: a set of function keys can be enabled with a toggle, and they can be used to execute a statement and stop at certain points. Stop points can be set for each function, or can be selected from the first or next line of any function, the current or lower function, or the calling function. As the execution proceeds, the stack and the currently executing function are shown in two separate windows. However, the results of executing the function lines are not shown, unless trace vectors are used.

## Problems in the Beta Release

- It is difficult to import or export an APLIWIN workspace to or from another APL system, including even SHARP APL on the mainframe. The *WSIS* workspace transfer mechanism should be available for this. I hacked a version taken from SHARP APL mainframe, in order to test some applications, but a proper job should be done here. Eric Iverson told me that a transfer facility will be included in the April release.
- The library name convention was taken from the original SHARP APL/PC. It associates PC drive letter with file account numbers. DOS path names are not allowed. This convention is old-fashioned and impractical. APL should instead adopt the convention of the underlying file system.
- The DOS file quad-functions signal domain error for a wide range of invalid arguments. It is sometimes difficult to find the real problem. More specific error reports would be useful.
- The non-TrueType font required for Windows 3.0 was missing from the disk I received. TrueType under Windows 3.1 provides a much better font support anyway, so I can only recommend the upgrade. Eric Iverson told me that the release version will be announced for Windows 3.1 only.
- I encountered a few Unrecoverable Application Errors under Windows 3.0, but none under 3.1.

- Besides the few differences from the mainframe language that are mentioned in the documentation, dyadic `⊔hold` is missing. These differences are not critical to single-user PC applications, but could be a problem if application code is to be taken unchanged from the mainframe.

## Comparison with the Mainframe Speed

I've tried to measure the speed of APLIWIN on my PC (66 MHz 486 DX2) and compare it to the speed of SAPL on a mainframe (the Risk Management Software RMX system). I used my *AIDA* workspace (an APL-like interpreter in APL) as the benchmark. It is very CPU intensive, and also uses files, but no screen I/O.

APLIWIN delivers about the equivalent of 3 mainframe CPU units per second (a SAPL CPU unit is one second of CPU time on an IBM 370/158). The RMX mainframe APL delivers about 75 CPU units per second under normal user load. This enables one to decide which applications could be ported; many applications typical of SHARP APL are clearly not candidates, since their interactive transactions (which consume 200 CPU units) typically take 2-3 seconds on the mainframe, but would run for more than a minute on the PC.

It would be interesting to compare APLIWIN's speed to other PC APLs, and to *Dyalog APL/W* with the *SHARP APL-to-Dyalog translator* from Insight Systems ApS. However, such comparison are beyond the scope of this *First Look*.

Martin Gfeller  
Reuters Risk Management Software  
Kleinstrasse 6  
CH-8008 Zurich  
Switzerland

# APLIWin and JWin

*reviewed by Jonathan Barman*

Iverson Software Inc. have released APL and J as Windows products. The Windows interface is identical in the two products, so if you have mastered it in APL you can immediately switch to J and run the equivalent code. The J disk comes with a DOS version as well.

## Getting Started

Installation was easy. I copied the disks, one for APL and one for J, into appropriate directories and was up and running almost immediately. On launching APL I got a message telling me that I had not installed the APL font. The standard Windows font installation soon put that right. These are full Windows programs, and you get the standard message if you try and run them in DOS. The icons included in the .EXE files are plain and ordinary, which suits me fine.

The APL font is very acceptable. The only difference between the ISIAPL font and Adrian Smith's APL2741 font is that the alphabetic characters are upright rather than italic. Also, there are no line drawing characters. The following is an example of a little function from the *isjwin* workspace in both fonts.

```

▽ r+id wdg r
[1] r+,>1+,(r[;⊞ic]ε<id)≠r
▽

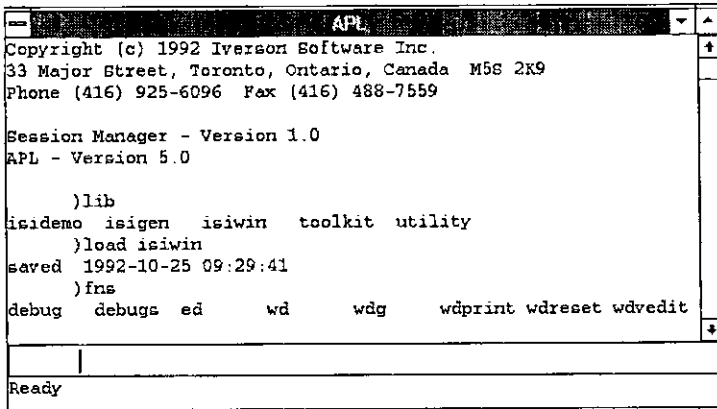
▽ r+id wdg r
[1] r+,>1+,(r[;⊞IO]ε<id)≠r
▽

```

On the screen the ISIAPL font looks cleaner with a much sharper definition at 10 point size. On paper there is not a lot of difference.

The session manager in both APL and J is identical, and appears to be a straight copy of the DOS session manager moved into a Windows environment. There is an execution line at the bottom of the screen, and the top part displays the results. The initial screen comes up filling up half the screen horizontally. I usually find that I need to change this shape, but there are no .INI files, so the programs do not remember the shape on exit. The following example of the APL

session shows the workspaces available and the functions in the *isiwin* workspace which we will be exploring later.



```

APL
Copyright (c) 1992 Iversion Software Inc.
33 Major Street, Toronto, Ontario, Canada M5S 2K9
Phone (416) 925-6096 Fax (416) 488-7559

Session Manager - Version 1.0
APL - Version 5.0

)lib
isidemo isigen isiwin toolkit utility
)load isiwin
saved 1992-10-25 09:29:41
)fns
debug debugs ed wd wdg wdprint wdreset wdvedit
Ready
  
```

I am used to the IBM, APL\*PLUS or Dyalog APL style of session managers, where one can type all over the screen to edit and execute anything displayed. It does not take long to get used to the ISI way of doing things, but having to cut and paste text down to the execution line seems to take a fraction longer. You can use the arrow keys to scroll up and down the actual commands that have been typed, but you cannot see them all to select the one you want.

The size of the window does not alter the way in which variables are displayed, which is purely controlled by  $\square p w$ . I have not found an equivalent in J, and expressions like  $1.1000$  wrap after some arbitrary number of characters. Expressions resulting in a wide character matrix, for example  $1":|:(4\#10)\#:1.2000$  (in APL  $1\ 0\ \uparrow(4\rho 10)\top 12000$ , except J requires  $|:$  or  $\diamond$ ), causes the screen to flash multiple times and eventually you get eight lines of output with a ragged right margin. The only way to see the right margin is to place the cursor on a line, press the Home key and then the left arrow key. It is most annoying not having a scroll bar along the bottom of the window so that you can see where you are in the wide display.

There is no menu bar along the top of the window, so none of the standard Windows features are included, such as changing the font size, or printing the session log. The standard cut and paste keys work in the normal way. With J this feature is most useful as it is easy to develop scripts in a Notepad window displayed along with the J session. Help is available with F1, which is reasonably full with APL. There is no list of all the APL primitive functions and operators;

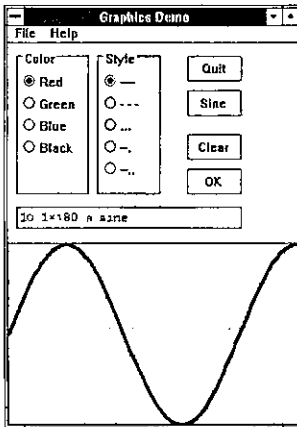
you need to buy the manual for that. The APL keyboard help is nicely done. The overstruck characters are distributed in a way with which I was not familiar, but otherwise it was easy to use. I only found by chance that O-U-T exit from `⎕` was in fact Ctrl-C. Obvious really, if I had stopped for a moment to think about it.

It is not possible to run multiple sessions of either program. Attempts to do so result in a check box appearing telling you that APL (or J) is already started.

## Windows Programming Example

Windows are created by running a system function `⎕wd` (111:0 in J) on a text vector right argument containing window programming statements. Each statement is separated with a semicolon, and "whitespace" of spaces, returns, tabs etc. are ignored, so you can edit a text vector with a normal editor. There are no symbols, and comments are allowed with the `rem` statement, so the result looks like some terrible form of Basic.

The APL documentation suggests that you run `gfx` in the `isidemo` workspace and become familiar with the application, so let us start with that.



I clicked on the Sine button, and then clicked on the OK button to get the sine wave at the bottom of the window. The `gfx` function looks quite simple:



```

      ▽ gfx;ap;aprun;apwtwp;apwr;apid;[]io
[1]  []io+0 ◊ aprun+1 ◊ ap+'gfx' ◊ gfxinit
[2]  lw:apwr+wd gfxwtp
[3]  +(ap▽apid+ap,'id' wdg apwr)/ln ◊ +(3*[]nc apid)/lb
[4]  lx:apid ◊ →aprun×lw
[5]  ln:→(3=[]nc apid+ap,3+'*type' wdg apwr)/lx
[6]  lb:→(←' 3 fkey'▽'*type' wdg apwr)/lm ◊ →lw-wd 'beep 3;'
[7]  lm:→lw-wd 'mb ','ap',' ','apid,' function not defined.'
mb_!conexclamation mb_ok;
      ▽

```

The *gfxinit* function which is called on line [1] contains a single statement *-wd gfxwtp* which runs the *wd* function on a global in the workspace. *wd* is a cover function for *[]wd* with a bit of error trapping. *gfxwtp* is a text vector containing the windows 'program', which defines all the push buttons, radio buttons, input area and graphic area. *→* throws away the result from *wd*.

Line [2] runs *gfxwtp* which sets the default input focus and waits for the user to do something. This time the result from *wd* is saved in *apwr*. Results from *wd* are a 2-column matrix of boxed character vectors. *[]wd* actually returns a simple character vector, but the cover function converts this into the desired format. In J the result of *11! : 0* is in the correct format, so no cover function is needed. When I clicked on the Sine button the following was returned:

```

*type    5 button
*parent  gd
*id      sine
red      1
solid    1
exp

```

The items in column 1 prefixed by a \* are system results; the remaining are the data from radio buttons and input boxes. *\*type* is always the first row and shows how the wait was ended. As I clicked a button I got the result 5 *button*. The 5 can be used as a quick way of checking the type, as each of the possible types are coded 1 to 9. *\*parent* gives the id of the parent window. *\*id* is the id of the child window which was active. The remaining lines show the data in the other child windows. The *exp* child window contained nothing.

On line [3] the child window *id* is extracted from the result by the *wdg* function. If the *id* is not 'gfx' but is a name of a function in the workspace then it is executed on line [4] and we go round the loop again. In my case the *gfxsine* function was executed, which contains the single statement *-wd 'csel exp;ctext "10.1×180 A sine";'*. The first statement *csel exp* selects the child window with the id of *exp*, and then *ctext* inserts the text into that window.

If the *\*id* is not the name of a function in the workspace, then something else has happened. Lines [6] and [7] sort this out. If you press the Esc key, or close the window with Alt-F4, then the *\*id* row of the result contains text telling you what happened. If that fails, then line [5] executes a function based on one of the 9 return types.

The guts of the program is in the *gfxok* function, which can be called if you click on the OK button, or if you press Enter in the child window where you enter your expression (*exp*):

```

▽ gfxok:[]trap;c;p;d
[1]  c+,(('red'>'green'>'blue'>'black')capvr[:0])/gfxcolors
[2]  p+,(('solid'>'dash'>'dot'>'dashd1'>'dashd2')capvr[:0])/gfxpens
[3]  []trap+''0 c []trap+''''o→le ◊ d→plot,exp' wdg apvr ◊ []trap+''
[4]  →0→wd 'cset gfx;grgb '.c.';gpen '.p.';glines ',d.';gshow;'
[5]  le:wd 'mb "Graphics Demo Message" "Not numeric
vector."'|mb_iconexclamation mb_ok;'
▽

```

Lines [1] and [2] get the colours and line specifications from the radio button settings. Line [3] sets up error trapping and then executes the statement entered in the *exp* child window. Line [4] selects the graphics child window, sets the colours with *grgb* statement, the pen line with *gpen*, plots with *glines*, and displays the result with *gshow*. Line [5] displays a message box with the *mb* statement, which has four parameters specifying the title, the text to be displayed, an exclamation mark icon and OK pushbutton. *mb* does its own show and wait, so you do not have to code these explicitly. Executing line [5] on its own results in:



and *wd* returns:

```

*type    2 nowait
*mb      OK

```

## GUI Programming

The *gfx* illustrates rather well the process that is necessary to program Windows in ISI APL or J. First you have to create the text vector that specifies the window 'program', which details all the child windows, radio buttons etc. that you need. Then you write a function which displays the window and waits for the user to do something. On returning from the wait all information in the window is returned to you, and you have to unpick the result to decide what to do next. Obviously you do not have to follow the style of programming adopted for *gfx*, which relies on  $\pm$  to avoid the multiple tests and branches that would otherwise be necessary.

ISI have provided a workspace *isigen* which enables you to write an application using exactly the same process as *gfx*. The *describe* variable in the workspace gives the step by step process required:

*gen* is a generic application. Build your application by making a copy of the *gen* application and adding graphical objects and the functions they invoke.

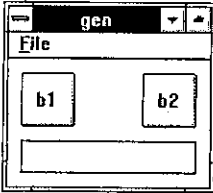
0. *gen*  $\mathbf{A}$  run *gen* and become familiar with it
1. *wsid nap* — chose a name for your application
3. *apcreate 'nap'*  $\mathbf{A}$  rename *gen* functions, variables and references
4. *nap*  $\mathbf{A}$  run new application
5. *edit napwp* to add a new button with an id of test
6. *nap*  $\mathbf{A}$  run application and press new test button
7. define *naptest* function to do something when test is pressed
8. *apvr* variable is the result of the wait
9. add controls and functions, add initialization code to *napinit*, and add cleanup code to *napclose*
10. look at *gfx* functions in *isidemo* as examples

The *gen* application has 4 functions and 2 variables:

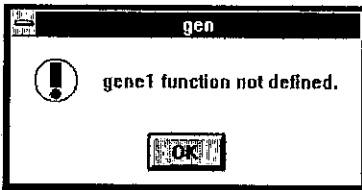
<i>gen</i>	- main function and the 'message loop'
<i>geninit</i>	- initialization and execution of <i>genwp</i>
<i>gencancel</i>	- process ESC key — <i>pclose</i> window and call <i>genclose</i>
<i>genclose</i>	- process <i>close</i> — cleanup and exit
<i>genwp</i>	- <i>wp</i> run in <i>geninit</i> to display application window
<i>genwtp</i>	- <i>wp</i> run at top of message loop — must end with wait

In spite of my almost illogical dislike of  $\pm$ , this is actually a good way to get a little application up and running quickly.

Running *gen* displays the following window:



Pressing the Enter key in the edit box displays a message box which shows that the edit window was called *e1*. Line [7] of the *gen* function was executed, which is identical to line [7] of the *gfx* function shown above.



The contents of *genwp* show how the window was constructed:

```
pc gen;

menupop "&File";
  menu open "&Open";
  menu cancel E&xit;
menupopz;

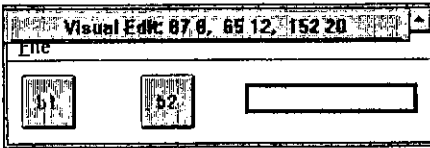
xywh 5 5 20 20;
cc b1 button;

xywh 50 5 20 20;
cc b2 button;

xywh 5 30 65 12;
cc e1 edit ws_border;
cfocus;
pas 5 5;
pshow;
```

The Windows program statement *pc* stands for Parent Create, and takes a parameter of the name of the parent window. *menupop* defines a single popup menu to go on the top of the window called File with the F as the accelerator key, and is terminated by *menupopz*. A 2-item pull-down menu has been defined under File. To define a button you have to first specify a rectangle with *xywh* and then specify a child window with the *cc* (Child Create) with parameters giving the id and the class of window. In this case *b1* and *b2* are buttons, and *e1* is an edit window with a border. *cfocus* makes the edit box the current window. *pas 5 5* adjusts the size of the parent window so that there is a border of 5 units around the child windows. *pshow* displays the window.

The *wedit* function allows editing of the positions of the child windows, once you have created a text vector specifying the parent window and its controls. I ran *a+wedit genwp* and then pulled the edit window to be alongside the buttons, resulting in the following:



As *genwp* contains a *pas* statement the parent window has been automatically resized to accommodate the new layout. This is pretty crude stuff compared with proper Windows development programs, but at least it enables you to dispense with typing in funny numbers for the positions.

## Window Facilities

As you can see from the examples, you are completely dependent on the facilities that ISI provide in their Windows programming language. There are 62 commands which seem to cover most of the relatively simple things that one might want to do, but by no means cover all of the amazing possibilities that a C programmer has at his disposal, nor the things that appear to be possible in Visual Basic.

The only classes of child windows that can be created are Button, Combobox, Edit, Listbox, Static and Isigraph. Classes are defined in more detail with Styles, for example a pushbutton can be defined as having the style of *bs\_autocheckbox*, or *bs\_groupbox*. There are general purpose styles, such as *ws\_border*, which can be applied to any of the classes of child windows.

Whilst getting to grips with the facilities available, I found the *winexec* command useful, mainly for dumping text and programs from the workspace into Write. A function is supplied to do the job:

```

∇ wdprint d;b
[1]  d+(1+b+d=CR)/d+,d,CR
[2]  d[(1+/b)+b/10b]+LF
[3]  -(⎕toascii d) ⎕hfwrite 'temp.wri'
[4]  -wd 'winexec "write.exe temp.wri";'
∇

```

The general technique could be used to provide a way of communicating with other Windows programs, even though having to write a file and then starting another copy of a program is slow and rather crude.

## Comparison with Dyalog APL

Dyalog APL/W was reviewed by Duncan Pearson in Vector 9.2, October 1992. Duncan created a number base converter to demonstrate the facilities available, which seemed at first sight to be very similar to those in ISI APL and J, so I will follow Duncan's steps. If you get out your copy of Vector 9.2, and turn to page 58, we will start at the top of the page.

To use the *isigen* facilities we have to load the workspace and set up the application:

```

)load isigen
saved 1992-10-25 19:19:57
)wsid bconv
was isigen
apcreate 'bconv'

```

I will modify the *bconvinit* and *wdvedit* functions to add *pas*, *pcenter* and *pshow* commands to the end of the *bconvwp* variable, so that each stage of the process can be seen. Normally these commands are added at the end of the command variable.

```

∇ bconvinit
[1]  -wd bconvwp,'pas 5 5; pcenter; pshow;'
∇

```

```

▽ wp+wdvedit wp;a
[1] wp+wd 'vedit;',wp,a+'pas 5 5; pcenter; pshow;'
[2] 'cancel' □signal(' 8 cancel'▽'*type' wdg wp)/8
[3] wp+(-pa)+'*vedit' wdg wp
▽

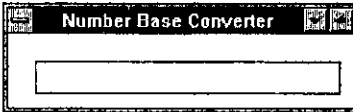
```

Now we can create the initial window with the initial edit box, and use *wdvedit* to make it look right:

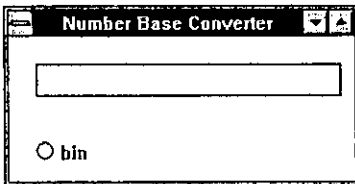
```

bconvvp
pc bconv; pn "Number Base Converter";
xywh 10 10 113 12 ; cc exp edit ws_border es_autohscroll;
bconv

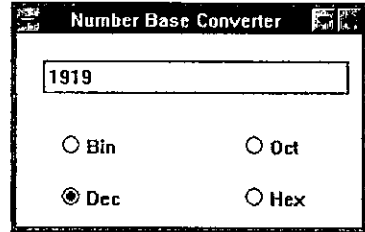
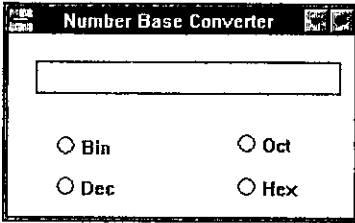
```



Using the same technique the statement `xywh 10 36 37 12; cc bin button bs_autoradiobutton;` was added for the Bin radio button, giving:



After a bit of messing around with *wdvedit* I got the window looking more or less like the picture at the bottom of page 59:



This involved using the `ed` utility, which is a function using the built in Window facilities to allow a simple Notepad type of editing:

```

edit
pc bconv; pn "Number Base Converter";
xywh 10 10 113 12 ; cc exp edit ws_border es_autohscroll;
xywh 17 35 37 12 ; cc Bin button bs_autoradiobutton;
xywh 85 35 37 12 ; cci Oct group;
xywh 17 54 37 12 ; cci Dec; ccheck 1;
xywh 85 54 37 12 ; cci Hex;

```

The next step is to write the function `bconvexp` to actually carry out the conversion. Here I met my first difficulty. I could not work out how to make a click on a radio button satisfy the wait so that one of my functions could be run to re-calculate the value in the edit box. It would seem to be necessary to add buttons such as Recalculate and Close in order to make this happen. The code for actually doing the conversion is much the same as Duncan's. I added a global variable to hold the old base, which is initialised in `bconvinit`.

```

V bconvexp; base; text; num; conv
[1] A Convert to base
[2] A Get the new base required
[3] base+({'bin'>'oct'>'dec'>'hex'})c apwr[;0])/ 2 8 10 16
[4] A Get the characters entered
[5] text+'exp' wdg apwr
[6] A Conversion string
[7] conv+'0123456789ABCDEF'
[8] A Convert from old base
[9] num+bconvold|conv|text

```

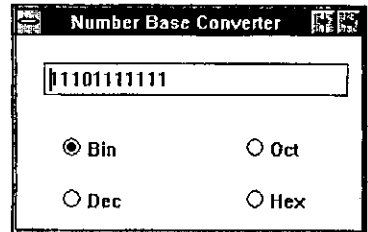
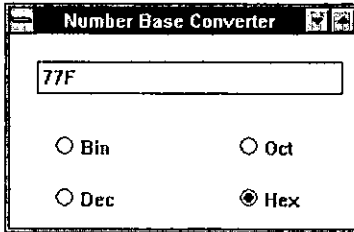


```

[10] A Convert to new base
[11] text+conv[([base*1+num)pbase)Tnum]
[12] A Display text
[13] -wd 'csel exp;text ',text,';'
[14] A Save base for next time
[15] bconvold+base
  ▽

```

Finally, we can display the same results as shown on page 62:



For each of the above results, I had to click on the radio button, then click on the edit window, then press the Enter key to get the number to change. There are a lot of functions available that I have not investigated, so perhaps there is a way of getting this to work more like Duncan's Dyalog code.

## Conclusion

This is a simple interface to Windows which enables one to produce simple applications quite quickly. I was a little surprised at the rather pedestrian way in which the interface has been implemented. I had expected that there would be considerable use made of boxed arrays to pass commands into `□wd`, rather than a character vector. Developing applications with few of the sizes and positions of child windows hard coded, and lots of little functions for every task, would require a lot of work catenating bits to a character vector that would eventually be executed by `□wd`.

The only documentation that I had was the Windows help text. I printed it all out, so that I could have it by me whilst developing the above, and it has taken me about 2 days to write this review. I think that this shows that the interface is easy to use.

## APL — New Release

*reviewed by Jon Sandles*

This new release of APL comes in a plain white sleeve and although there is no documentation available, I do not believe it will ever be commercially available. Believe it or not I first heard of this on the television. Well how does it shape up against its competitors? ... all the standard features are there. A full heady riff is developed slowly and steadily with a nod and a wink to past masters of this genre. The pulsating beat never lets up and by the end you are left drained of all emotion. Of all APL releases this well could be the first to break the top 40. Criticisms of the 12" disk format (I could not find any commercially available 12" drives for PC's) will abound, and the black vinyl format is rumoured to be obsolete. That aside Manugistics, Dyalog, MicroAPL et al could learn a lot by the fulsome groove. I also wonder — where did they get that silly name.

The influences are worn proudly, from Joy Division to the KLF with snatches of old — Floyd, Genesis and even Tangerine Dream. Will these people get better? — I hope so because few records these days sound as fresh as this. (Note: as usual with APL products the release date of this record has been put back — it is on the *Beyond* label and the track is called '*the calling*'.)

---

Here are some extracts from a promotional leaflet which your editor recently received ... readers are invited to offer suggestions as to just what "Vector" might be. Anything sufficiently silly will probably get printed in the next issue ....

- **Vector** is clean quick and easy to apply
- **Vector** is cost effective and labour saving
- **Vector** is ready to use and does not require dilution
- **Vector** does not stain the fleece
- **Vector** is a non organophosphorus product
- **Vector** minimises mothering-up problems
- **Vector** eliminates the risk of chilling lambs

### Protection of Operators

- **WEAR PROTECTIVE CLOTHING, RUBBER GLOVES AND BOOTS** when applying the product. Use in a well ventilated area.

---

# RECENT MEETINGS

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

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

---

# APL-landing to Kronstadt Island (an experience of APL operation)

*from Pavel & Oleg Luksha*

Key-words: . . . APL, Russia, Kronstadt, IBM PC XT/286 with no hard disk,  
TryAPL2, Per Gjerlov.

## **A Bit of History**

We have got this precious experience you will read about in the cold and windy November heart of Russian Navy locations.

Perhaps we can tell nothing if you have never heard about Kronstadt. Its tale begins with Sweden warriors who have occupied this island once upon a XIV century. Then there was a kind of ping-pong game: it was occupied by Russians, then by Swedes again, then by Russians again. Finally it was reserved by Russian czar, Peter The Great, who has built a fort there and said his famous words: "From here we'll threaten Swedish labour, for fear to be our haughty neighbour!". As the translation is ours, to say it in ordinary language: "This island belongs to me!" Then there were some wars, some revolutions, until we had the year of 1992 coming. This historical (with no doubt) year APL'92 was held in St.Petersburg. When it was held, some participants had clearly decided in addition to their sightseeing of Kronstadt to visit the school of that beautiful sea-town.

## **How has this story begun?**

All persons who paid that noble visit were veterans of APL movement. They found a Kronstadt school having several IBM PCs, but no APL at all! Everything they could do was Erkki Juvonnen's courses plus Per Gjerlov's visit. They saved up money enough to pay for whole of Erkki's and half of Per's. The only problem was to interpret what Per would say (as they had only several persons speaking English and none of them could understand APL terminology).

So he had invited us and we came in a moment. Nick Puntikov has taken us to Kronstadt immediately.

Everybody was excited with Per's visit, especially island teacher Mr. Spiegel. He (i.e. Spiegel) made himself an engine of the course. Thanks to him Per and we

could find an accommodation on the very island. So we found a place to stay at a nice ancient hotel, which is known there as "Three Bananas". A small apartment contained a long and narrow room with furniture of style and age of 50ths. But we were fond of our hotel (at least we hadn't had to sleep under an open sky). And all during the days we had splendid time and luxurious parties at Spiegel.

### What is the Problem?

The school we taught in was not an ordinary one. Pupils were tested thoroughly to get in there. They're now taught of different kinds of subjects with the newly-designed courses.

The other important thing is that they are studying Junior Achievement course, which is said to be "the oldest, largest and fastest growing" economics program in the world, so more than 1.5M students participate in its different courses each year. This subject (i.e. economics) is very new and actual for the former USSR republics. Suddenly we were thrown into the cold water of wild business, so we had to learn how to swim in there. Some study it by their own skin, others take course like JA. It includes theory of marketing and management (with the basic economics), business English and a computer program to practise the lessons for better mastering.

This program was designed by Harvard Associates which develops different educational software (including JA software also). The program simulates an economic situation and lets a student decide what to do. Then it returns the results of the decision and lets decide once more. It is really powerful for educating, but there are the following features it does not have: graphical representation of results exceeded, mathematical and statistical tools.

These last are those which APL is best for. So the idea was if APL could be used for economics. We were able to prove it could!

Though economics seems to be social science, it has a great number of mathematical tools to use (from tax and profit formulas to various analyzing systems) to get the results.

APL is very similar to jazz; it gives you all the possibilities to improvise. Actually, we had no strict plan to follow; but that what we were doing was double brain-attack. First was in the evening when we were discussing an economic (or any other) problem ourselves; next day we were bringing the results into the class and discuss again in there. Using this method we could work out an efficient way of graph drawing (even with those graphical primitives of TryAPL2).

So the most excited thing was that (though we had it taught for 2 days only) pupils WERE interested in APL. They constantly asked if they would continue their lessons.

### The Rest

All the rest is covered with a thin mist. The mist is thin because recently the author received a letter from Kronstadt. They say Per's visit was really enjoyable.

In another letter, received from Per himself, who successfully reached Denmark, it is said that Kronstadt would be his best memory of the year. Same is for us.

### The Results

Actually, the work is not finished yet. Per is going to visit Kronstadt for another session. We're going to join him.

Now we could learn the difference between APL educational programs. It should be noticed that I-APL proved that it was really the best for LEARNING one. It has ready-made self-introducing WS, mathematical tools Wses, graph plotting WS, music and memory operations Wses etc. This is what IBM can use to show the possibilities and power of APL2.

But we enjoy both I-APL & TryAPL2 and hope they both will prove their great use for education, even economics.

In addition, we were interested in JA course ourselves, and we are going to make APL-accompaniment for this program, like APL-idioms & APL-exercises. We ask anyone who knows of using APL for this or other economic courses and can help us in this regard, please contact us at: [luksha@market.avecs.obninsk.su](mailto:luksha@market.avecs.obninsk.su)

# An Implementation of J

## Roger Hui's presentation to the British APL Association on 12 February 1993

*transcribed by Anthony Camacho*

It is a pleasure to be in London again and an honour to be invited to speak to this audience. I am going to talk about **An Implementation of J**. Please interrupt me at any time if you have questions.

### What is J?

J is a dialect of APL based on Ken Iverson's work over the last forty years. It uses the standard ASCII symbols and therefore does not require the special keyboards, special displays, special printers, special editors and so on that previous APLs did.

It has facilities which enable functional programming. It is freely available and runs on many machines including:

- Sun 3
- SPARC
- Silicon Graphics
- Mips
- Next
- RS 6000
- Vax
- PC
- Macintosh
- Archimedes

and others.

It is written in C and is portable. The source code is available. It uses standard facilities. For example it uses STDIN and STDOUT for dialogue. It uses the C library functions malloc and free for memory management and it provides access to host files or native files.

The following dialogues give a taste of the system. The lines that are indented are those I typed; the lines that begin at the margin the system's responses.

```

a =. 1 2 3 4 5 6
sum =. +/
sum a
21
mean =. sum % #
mean a
3.5
report =. i. 2 3 4
report
0 1 2 3
4 5 6 7
8 9 10 11

12 13 14 15
16 17 18 19
20 21 22 23

```

The first sentence says 'a' is a list of six numbers.

'sum' is a verb or function which computes the sum. The sum of 'a' is 21. We call 'sum' a verb because it applies to a noun to produce another noun. The symbol slash (/) is an adverb because it applies to a verb, in this case plus (+), to produce another verb. 'mean' is the sum divided by the number of items. The mean of 'a' is 3.5.

Verbs in J apply to elements, lists, tables and reports. For example suppose 'report' is the revenues for two departments over three countries over four quarters. The mean over the departments is simply 'mean report'.

```

mean report
6 7 8 9
10 11 12 13
14 15 16 17

mean "1 report
1.5 5.5 9.5
13.5 17.5 21.5

mean "2 report
4 5 6 7
16 17 18 19

```



```

      mean "3 report
    6  7  8  9
   10 11 12 13
   14 15 16 17

```

'mean' over the four quarters is 'mean' applied to the list of rank one objects in report and 'mean' over the three countries is 'mean' applied to the tables of rank two objects in 'report'. 'mean' applied to the rank three objects in 'report' is the same as 'mean report' because report only has rank three.

The symbol double quote used here is the 'rank' conjunction. It is dyadic and applies to a verb left argument and a noun right argument to produce a verb.

```

      ^ 2 3 4
    7.38906 20.0855 54.5982
      1 2 3^2 3 4
    1 8 81

```

```

      square =. ^&2
      square 1 2 3 4
    1 4 9 16

```

```

      antilog =. 10&^
      antilog 1 0.699 _1
    10 5.00035 0.1

```

```

      ss =. +/&(^&2)
      ss"1 report
    14 126 366
    734 1230 1854

```

The symbol hat (^) denotes a verb and like other verbs it has a monadic and dyadic meaning. The monadic meaning is exponential. The dyadic meaning is exponentiation or power.

Now if you fix one of the arguments of a verb you get a different verb. For example 'square' is power with a fixed right argument of two. Ant 1 log is power with a fixed left argument of ten.

The symbol ampersand denotes a conjunction. If one argument is a noun it does fixing or currying; if both arguments are verbs it does composition. For example sum of squares is sum composed with square.

Like all verbs, these verbs that are derived from conjunctions apply to lists, tables and reports and are in the domain of conjunctions.

## Nouns (arrays)

In the implementation the fundamental structure is the APL array, by which I mean the C structure capital a (A), which has the following parts:

The type

Reference count

Number of atoms or elements in the array

The rank

The shape or dimensions (the rank gives you the number of elements in the shape)

The value — the elements of the array in ravelled or 'row major' order

```
typedef long I;
typedef struct {I t,c,n,r,s[1];} * A;
```

t	type
c	reference count
n	number of atoms
r	rank
s ...	shape
v ...	atoms of the ravelled array (row major order)

All objects, whether numeric, literal or boxed, whether noun, verb, adverb conjunction or punctuation are represented by this structure. For example the string 'cogito ergo sum' is represented like this:

'Cogito, ergo sum.'

CHAR	1	17	1	17	CogI	to,	ergo	sum	.
t	c	n	r	s	v	v	v	v	v

The type is character. There are seventeen elements. The rank is one and the shape seventeen. The value is the seventeen characters in the string held in one byte per element or four bytes per word.

The number 1.61803 is represented as follows:

1.61803

FL	1	1	0	1.61803
----	---	---	---	---------

The type is floating point. There is one element. The rank is zero so there is no shape and there are two words per element in the value.

Questioner from the audience: *What is the reference count?*

The reference count is the number of times this object is used. I won't go into that. It has to do with the internal workings of the interpreter and is not very interesting.

The report we saw earlier is represented as follows:

i. 2 3 4

INT	1	24	3	2	3	4	0	1	...
				...	21	22	23		

Type is integer. There are twenty four elements. The rank is three. The shape is 2 3 4 and the value is the integers from 0 to 23 each stored in four bytes.

I said before that not only nouns but also verbs, adverbs and so forth are represented by this structure; so for verbs, adverbs and so on the type would be verb adverb and so on, but the value would interpreted according to the following template or structure denoted by the C structure `V` having these parts:

```
typedef A(*AF)();
typedef struct {AF f1,f2;A f,g,h;
               I mr,lr,rr;C id;} V;
```

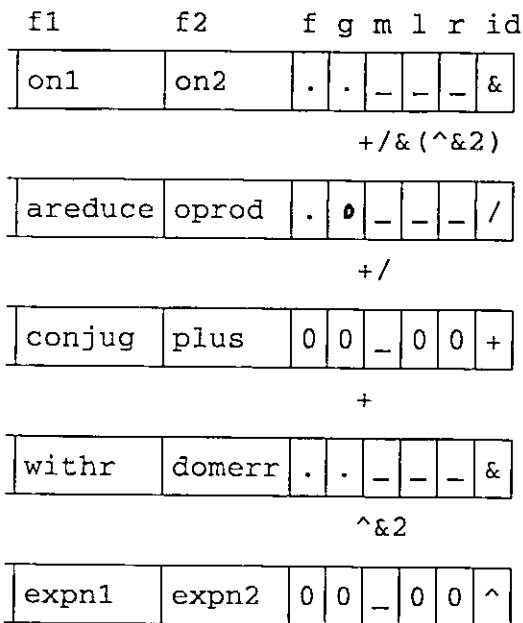
f1	monad
f2	dyad
f	1st operator argument
g	2nd operator argument
h	3rd operator argument
mr	monadic rank
lr	left rank
rr	right rank
id	identification (byte)

If a verb has rank `R` that means it is defined on arrays of rank `R` or less and the extension of that verb to arrays of higher rank is the same as for all other verbs.

Questioner from the audience: *What are the first lines on the slide?*

Oh that is really for C hackers; it doesn't really matter very much. The top line is defining a type called 'AF' and that's a function that returns an array result. I put it here because I use 'AF' in the second line. The second line says that 'f1' and 'f2' are of the C type 'function' returning an array result and 'f', 'g' and 'h' are of the type 'APL array' and 'mr', 'lr' and 'rr' are the C type 'integer' and 'id' is a C type capital c (c). I haven't shown the definitions of I and C but they are just integer and character.

To give you a better idea of what all this means: the verb 'sum of squares' we saw earlier would be represented as follows.



I haven't shown the rank, shape, reference count, number of elements and so forth — I chopped it off because the interesting part is what is shown here.

For 'sum of squares' the root is composition whose symbol is ampersand (&) and whose C function is 'on1'. The dyad is the C function 'on2' and all the ranks are infinite. The operative arguments 'r' and 'g' are 'sum' and 'square', themselves represented similarly.

'sum' the symbol is slash (/). The monad is the C function 'areduce'. The dyad is the C function 'outerproduct' and there is only one operative argument, plus (+).

For 'plus' the symbol is plus (+). The monad is the C function 'conjugate'. The dyad is the C function 'plus' and there are no operative arguments because plus is primitive.

Back to 'square'; the symbol is ampersand (&). The monad is the C function 'withr'. The dyad is the C function 'domainerror'. The operative arguments are

'power' and '2', themselves represented similarly. '2' is a noun whose representation we've seen before. 'power' has the symbol hat ( $\wedge$ ). The monad is the C function 'exponent 1a11'. The dyad is the C function 'exponentia12' and there are no operative arguments because power is primitive.

So I think you can see how this can grow on and on to make more and more complex functions.

## Parsing

This parse table is the cornerstone of the interpreter.

```
typedef struct {I c[4];AF f;I b,e;} PT;

#define EDGE      (MARK+ASGN+LPAR)
#define NOTCONJ  (NOUN+VERB+ADV)

PT cases[] = {
EDGE,      VERB,      NOUN,      ANY,      monad, 1,2,
EDGE+NOTCONJ, VERB,      VERB,      NOUN,      monad, 2,3,
EDGE+NOTCONJ, NOUN,      VERB,      NOUN,      dyad, 1,3,
EDGE+NOTCONJ, NOUN+VERB, ADV,      ANY,      adv, 1,2,
EDGE+NOTCONJ, NOUN+VERB, CONJ,      NOUN+VERB, conj, 1,3,
EDGE+NOTCONJ, VERB,      VERB,      VERB,      forkv, 1,3,
EDGE,      VERB,      VERB,      ANY,      hookv, 1,2,
EDGE,      ADV+CONJ, RHS,      ADV+CONJ, formo, 1,3,
EDGE,      ADV+CONJ, ADV+CONJ, ANY,      formo, 1,2,
EDGE,      CONJ,      NOUN+VERB, ANY,      curry, 1,2,
EDGE,      NOUN+VERB, CONJ,      ANY,      curry, 1,2,
NAME+NOUN, ASGN,      RHS,      ANY,      is, 0,2,
LPAR,      RHS,      RPAR,      ANY,      punc, 0,2,
};
```

A sentence to be parsed is placed on a queue and as parsing proceeds, words are moved from the queue onto a stack. After each move the first four words on top of the stack are compared to these patterns. If they match a pattern then the action in this column is triggered and that action will be applied to the words indicated in the last two columns and the result of the application put on the stack in place of the matching items.

The implementation makes extensive use of macros, defined constants and type definitions. You have already seen some of them; for example the types A and V, the defined constants noun, adverb, conjunction and so forth.

The advantages of such usage is that it greatly augments the expressive power of C, it enforces uniformity and increases readability. For example, by 'an APL function' I mean a function that applies to array arguments and returns an array result. These macros encapsulate that convention. The macros 'f1' and 'f2' are for primitive APL functions and 'df1' and 'df2' for derived or non-primitive functions. The argument 'self' is an array whose monad or dyad is 'f'.

```
#define F1(f)  A f( w)A  w;
#define F2(f)  A f(a,w)A a,w;

#define DF1(f) A f( w,self)A  w,self;
#define DF2(f) A f(a,w,self)A a,w,self;
```

Using such definitions and macros the functions and programs in the implementation look like this. The monad itemised is defined in one sentence. Likewise the C implementation is one line. Notice the use of the 'f1' macro which says that this is a monadic function which applies to one array argument and returns an array result.

Dictionary:

, :y adds a single unit axis to y, making the shape 1, \$y.

C:

```
F1(lamin1){R reshape(over(one,
    shape(w)),w);}
```

The dyad 'laminat' is also specified in one sentence and again its C implementation is one line. The 'f2' macro indicates that this is a dyadic function which applies to two array arguments and returns an array result.

Dictionary:

An atomic argument in  $x, :y$  is first reshaped to the shape of the other (or to a list if the other argument is also atomic); the results are then itemized and catenated, as in  $(, :x), (, :y)$ .

C:

```
F2(lamin2){RZ(a&&w);
  R over(AR(a)?lamin1(a):a,
  AR(w)?lamin1(w):
  AR(a)?w:table(w));}
```

The conjunction 'ampersand' that we saw earlier is implemented as follows. As indicated previously if one argument is a noun and the other a verb then it does 'fixing' or 'currying'. If both arguments are verbs then it does composition and if both arguments are nouns then it signals domain error.

The functions 'on1', 'on2' and 'withr' that we saw earlier are defined thus.

```
static DF1(with1){DECLFG; R g2(fs,w,gs);}
static DF1(withr){DECLFG; R f2(w,gs,fs);}
static CS1(on1, f1(g1(w,gs),fs))
static CS2(on2, f2(g1(a,gs),g1(w,gs),fs))
F2(amp){
  RZ(a&&w);
  switch(CONJCASE(a,w)){
    case NN: ASSERT(0,EVDOMAIN);
    case NV: R CDERIV(CAMP,with1,0L, RMAXL,RMAXL,RMAXL);
    case VN: R CDERIV(CAMP,withr,0L, RMAXL,RMAXL,RMAXL);
    case VV: R CDERIV(CAMP,on1, on2,mr(w),mr(w),mr(w));
  })}
```



## Statistics

C Fns	793
Lines	4521
Average lines/fn	5.7
Min	1
Max	81
Median	1
One-liners	435

Lines	4521
+ / Line lengths	143481
Average chars/line	31.7
Min	1
Max	89
Median	28
One-character lines	293

440 of the 793 fns are APL functions

These statistics are derived from the J source code for version 6. As you can see the implementation consists of a large number of functions which are short, having short lines and following a well defined uniform interface. These are characteristics of an APL programming style.

This concludes the prepared part of my talk. Are there any questions?

*[Reporter's note: The question and answer exchanges below are summaries of what was said.]*

- Q Why should APLers use J?
- A Because it has no special characters, enables functional programming and for other reasons — why don't you invite Ken Iverson to come and talk to you about it?
- Q Can all APL functions be translated into J?
- A What I called an 'APL function' was defined as a function that processes arrays and gives an array result. This is not necessarily the meaning of APL function in APL\360. Translation from APL to J has to be done by hand.

- Q Does J have the concept of the workspace?
- A Yes. It is currently implemented in a workspace interchange form and future versions may use different representations.
- Q Can you tell us something about J's relationship with its operating environment?
- A It provides an interface to host or native files and also can call functions that follow the C calling convention. The details are to be found in appendix C of *An Implementation of J*. It is called the link-J interface.
- Q J has changed rapidly over the last few years: to what extent was this implementation style designed to make such change possible?
- A I had a slide of statistics at Copenhagen over two years ago. The figures were very similar to these. This programming style has actually evolved out of desperation because that was the only way I could keep up with Ken Iverson.
- Q Were you an APL programmer before you did C?
- A Yes; I was an APL programmer for about thirteen years before I did this. I knew APL from the outside before writing its inside.
- Q Would you have written the C like this without that experience?
- A Probably not. These statistics are one indication that this is an APL programming style effected in C.
- Q Are there any efficiency consequences; is this style of C slower than others?
- A No; no necessarily. Between Copenhagen statistics and these I did an extensive speed up without affecting the style or the statistics.
- Q Are there any limitations such as a maximum rank?
- A Yes, there is an artificial limit on rank of 127. There is also a limit on the number of elements in an array — as it is stored in four bytes there is a limit of two billion.
- Q What is 'EDGE' in the parse table?
- A 'EDGE' is 'marker', left parenthesis or assignment arrow.
- Q Please explain parsing again as I didn't follow.
- A Words are moved from the queue to a stack and the top of the stack is compared to the pattern in the parse table. Suppose we are parsing  $a = . 1$ . The queue will contain a marker ' $\{$ ' followed by  $a$ ,  $=$ , and  $1$ . We move  $1$  to the stack. There is no match so we move  $=$  to the stack; still no match. We move  $a$  to the stack. The stack now contains  $a = . 1$  and this matches line 12 of the parse table because RHS is any of noun, verb, adverb or conjunction, so we invoke the C function '1s' with the arguments indicated in the last two

- columns; the result is a 1 which replaces the  $a = . 1$  on the stack. We then proceed from there.
- Q So to change the way hook works you would change this parse table?
- A Yes. If you take out some lines from this table you get the APL\360 parsing rules. That's why hook and fork and so on are proper extensions to the APL\360 rules: we just took expressions which would have been errors and assigned meanings to them.
- Q What about currying?
- A Lets look at the pattern: if you have two adverbs in a row or two conjunctions in a row or if you have an adverb and a conjunction then that fits the pattern. An example is '+\' which is 'sum'. To define a scan like the APL scan all I need is '/\' — two adverbs in a row and that would be handled by this rule.
- Q Could the parse table really be used to parse APL?
- A It would have problems with anomalies such as semicolon bracket indexing and it couldn't do strands, but otherwise it would work.
- Q What about 1 space 2 space 3?
- A We consider that part of word formation rather than parsing. That is done before putting sentences in the queue.
- Q How is memory used?
- A I ma l l o c each little bit as I go, so how memory is used depends on how ma l l o c works.
- Q What about saving a workspace?
- A What I mean by a saved workspace is slightly different from other APLs. I just put each object in turn into a standard representation.
- Q Why does loading a workspace use more memory than the size of the workspace when on file?
- A The standard representation packs objects economically. On loading they are expanded to a form which is easy to handle — the form you have seen. Also the process of doing the conversion uses space.
- Q Between J and APL, which do you prefer?
- A I prefer J because I implemented it.
- Q Is J APL?
- A Yes — it is obvious that J is enhanced APL thinking.
- Q Which goes faster, J or APLIWIN?

A The windows code is the same for both and this takes most of the processing. I believe J is competitive on the rest of the timing.

Q What are your hopes for J?

A I have no ambitions in that regard.

Q Could someone with an APL background understand your C?

A Yes definitely. The source assumes the reader knows both J and C but the reader who knows C and not J or APL is under a much greater handicap than an APLer who doesn't know C.

Q How do conventional C programmers react to this?

A With horror!

Q Can you describe the workspace environment?

A It's perhaps a little misleading to describe the space where objects are held as a workspace, because all it is is space obtained from `malloc` and freed by `'free'` when no longer used. Again see appendix C of the book.

Q Do you have things like symbol tables as well?

A Yes, the symbol table is just another type of object with the type `'symbol table'` and it relates the name to the value. Either name or value could be used when the symbol table is referred to depending on what would be most convenient.

Q Can you have multiple symbol tables to avoid 'symbol table full' messages?

A Yes I do but not for that reason. I use multiple symbol tables to hold localised variable names.

Q On the J disk of source code version 6 there is a directory J41 as well as a directory J6. Why?

A The book is fully compatible with J version 4.1 but J version 6 is the latest version.

[Applause]

**Announcement:** The book *An Implementation of J* by R K W Hui, published by Iverson Software Inc, is available from ISI in Canada for \$90 plus \$20 for air mail and packing. These are US Dollars not Canadian. I-APL will get it for you for sixty pounds plus three pounds packing if you order from the enquiry address. It is not shown on the I-APL price list because sales do not justify the space it would take.

---

# GENERAL ARTICLES

This section of VECTOR is oriented towards readers who may neither know APL, nor be interested in learning it. However we hope you are curious about how, under the right conditions, such impressive results can emerge so quickly from APL programmers

---

# APL Experiences and MicroSoft's Visual Basic for Windows

*by Martyn Adams*

I am a Visual Basic programmer. Please stop laughing! I am in disguise you see, I am really an APL programmer but there are reasons why I am not programming in APL any more.

Seriously though — I will have to explain why the situation is as it is and hope that someone, particularly an APL vendor, will take a lesson or two from this article and address some of the problems I have encountered. I am/was a pretty good APL programmer in my time but market forces have changed things and I have had to adapt to the market place. Now I like to see myself as a sort of fifth columnist — examining the competition and feeding back the details for the benefit of my colleagues in the APL world.

I suspect you are like me, once you have got to grips with Iverson Notation other languages are just a verbal abuse of your time and energy.

So then, I will start by giving you a brief history (not long enough to be boring) that will put things into perspective and explain how I arrived where I am, and then I will delve into Visual Basic (VB) by running through a sample VB session. The session itself will create a simple Windows program and I will let you judge how easy it would be to program in APL for Windows — whichever APL that is.

Before I start though, a word of warning. During the last five or six years I have come across some pretty fundamental changes in software design and build methodologies. Two new buzz-phrases are Object Oriented Programming and Graphical User Interfaces (GUIs) like Windows. If you hear someone say that OOP is just like subroutines, and GUIs are more complicated than necessary — they are dangerously WRONG! I should know — it cost me.

I have seen experienced software developers ignore Windows and stick to DOS for as long as they could and then try to implement their old style monolithic programs under the Windows environment. It never works. The net result of my tolerance to such follies has cost me and my company at least thirty thousand pounds over the past year — and that is just through this simple lack of foresight and understanding by a programmer. Take heed, programming style has changed — and you must be aware of these changes and change with it!

The mentalities I have come across seem to break down into two types:

**Type 1** are the old established players who would not contemplate developing products under Windows until it had become something everyone was using. Now it is probably too late for them to make a credible jump and a wonderful opportunity has been lost for ever.

**Type 2** are the old established players who did make the jump but did not understand what Windows is all about. They did not understand OOP or GUIs or what they were really trying to achieve and consequently made a mess of the entire affair. Their attitude was one of blind indifference and even cynicism when it came to Windows.

Notice that it is mainly the 'old established players' which form this resistance movement. If you are one of those you could well find your future in programming rather difficult. You have been warned. I hope you are not amongst this crowd — if you believe you are, or might be, buy yourself a copy of VB and develop a little program in it. VB is not OOP — but it does not have to be as we will see.

OK then, history. A few years ago I was employed by a consultancy and we developed a pretty good system in APL. It ran, naturally, slower than a comparable system developed in C or PASCAL — but our development costs were extremely low and the speed at which we developed the system was phenomenal. We stayed streets ahead of the competition by improving the functionality of the system and our clients were quite prepared to buy top-end machines to offset those performance issues. This system ran under good old DOS.

The problems we encountered with this development culture were trivial at first, but grew until they became quite considerable. Firstly, we had to ship a complete copy of the APL interpreter with every copy of software, giving clients access to our original source code. Later we had a Run Time System, however we then had to negotiate discounts and distribution deals as a special case as we were not a common phenomenon in the APL community. Then the APL vendor decided to improve the usability of the interpreter at the expense of workspace and speed. The features we wanted were not included.

To top it all, we had clients purchasing machines with (the then) new EGA and later VGA screens and we had to wait six months before the upgrades could be properly reflected into the design of our software. Networks seemed to come as a complete surprise to APL language developers and we had to develop some tricky APL code to enable multiple concurrent access to our large databases.

It was also pretty apparent that the interpreter was not keeping up with the fast moving PC market. Fortunately APL is such a powerful language that you can code around most of the shortcomings in the language itself. But worse was the endemic APL culture — never mind the professionals trying to build and ship a professional product — let us help the learners. Of course there is nothing wrong with that, on the contrary, we need more APL introductions which are easy to use. But without the delivery of commercial systems how will APL ever become a respectable name in the software market?

Then came OS/2 closely followed by Windows. It took years before APL really caught up, and I am not sure it has yet. Look at the VB demonstration later in this article.

At the end of 1990 I left to develop my own 'New Generation EIS' software package using APL as a prototype. Although it generated a lot of interest and it was technically competent, it was never a viable product. APL still was not on Windows in a form in which I could develop and ship stand-alone products, and everyone was asking me for Windows software. So I had to switch to another language.

I chose a new 'Fourth Generation Language' from a new British supplier. Eighteen months later substantial technical issues built up until I could no longer sustain product development in this environment. Old fashioned concepts were applied to development which gave me no end of trouble with the Windows environment. Finally, the product was killed off with an over ambitious pricing structure.

So I had to switch again, but to what? My options, based on my own programming experience, were:

Microsoft C

Microsoft Visual Basic

Borland C++

Borland Turbo Pascal for Windows (TPW)

An APL for Windows

Well, I had tried Borland's C++ and TPW — and I was very impressed with TPW. It compiled like lightning and had a really well integrated development environment which enabled stepping through the code, examination of variables, error trapping, breakpoints, watches, stacks, back tracking, full access to the windows API, ability to build custom DLLs, etc. Even better debugging



than most APLs! No Run Time Systems to worry about, no licensing problems. Even when using the editor the source code is highlighted in colour according to the syntax. It is Object Oriented too.

MicroSoft's C at that time was like any other C product but worse. HUGE!!! The index for the documentation alone is over 500 pages long! I remembered back to the good old days when our development costs were low and the development times far better than the competition. "Not for me!" I thought.

APL? Well, there are one or two good APLs on the market which might have filled the bill. But I chose not to go that route for many reasons. I shall list some of them:

1. The additional cost of APL interpreters or Run Time Systems.
2. Interaction with some of the APL systems was not truly Windows like.
3. Insufficient flexibility with access to Windows APIs, DLLs etc.
4. Less APL expertise around than say, PASCAL, C or BASIC.
5. Cannot expect users to purchase and adapt the source code based on APL.
6. Past experiences.
7. Smaller user community.

So I took the pragmatic decision and from the list of languages I picked two. For product development we use Borland's PASCAL 7.0 (an enhanced version of Turbo Pascal for Windows), and for client customisation and other bespoke work we use MicroSoft's Visual Basic. They are both cheap, well integrated into the Windows environment, nearly as productive as APL, faster to run (well the PASCAL is anyway), future proof to a greater extent than APL and easier to market.

I hope that explains my position clearly and gives some real-life feedback to the APLers of this world.

### **Now to the Visual Basic Example**

I use VB2 Pro — its full title is Visual Basic Professional release 2.0. It only came out in January 1993 and costs about £350. Release 1 came out the year before. Installation is straightforward, a little time-consuming and a little space-consuming, mine is currently up to 15 Megabytes but I do not believe you need all of that.

It comes with all the stuff you would expect:

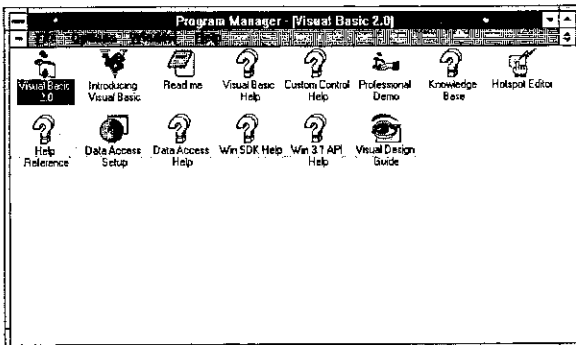
- a simple tutorial (noddy really)
- a professional's tutorial (simple really)
- example icons
- a context sensitive language help system
- a development environment
- examples of applications — including an icon editor.

and some useful extra goodies:

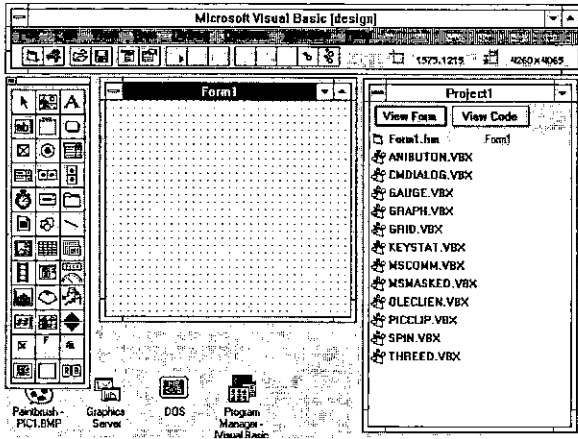
- a help file on general experiences and how to do things in VB
- a help file on the Windows API
- a help file on the Microsoft official Windows look and feel
- a help file on how to extend the VB windows controls (buttons, lists etc.)
- the source code for a product installation tool kit written in VB
- a help text compiler, together with a hot-spot editor for hypertext type stuff

Not bad?

So, for our example let us assume that we have installed VB2 and we are going to write a simple application which enables us to edit simple ASCII text files with full Windows look and feel — like the NotePad program that comes free but better. First, we open the Program Manager Visual Basic 2.0 program group and goggle at the icons.



Second, we double click on the Visual Basic 2.0 program icon to reveal four new windows. In the screen image I have minimised Program Manager, otherwise the screen gets a little cluttered.



At the top of the screen is the main VB menu window. With the screen you actually control the VB development session by adding new source files, forms (I will explain these in a moment), compile into an executable program, switch between projects, start/stop/trace program execution and set the default colours and such like.

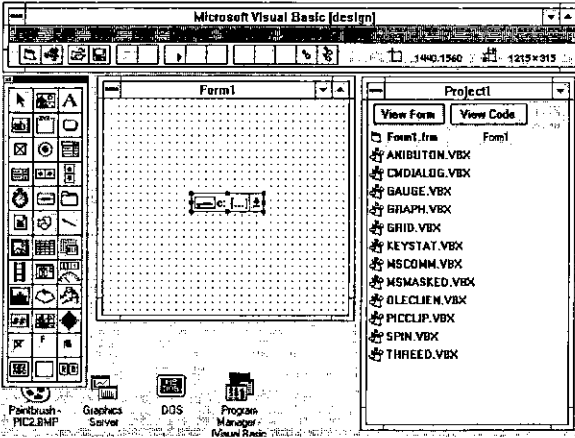
To the right is a window with a list of the program libraries and form files. Double click on these to start editing the code but as we have only just begun there is only one form called Form1. The VBX files are extensions to VB which enable 3D controls, multimedia, simple spreadsheet emulation, graphs, OLE, email, communications etc.

To the left is the tool box: you click on this to select the bits and pieces that go onto a form. Each one of these bits and pieces is called a 'control'. A control is something like a scrollable list, a data entry box, a place to put text, an icon, a button etc.

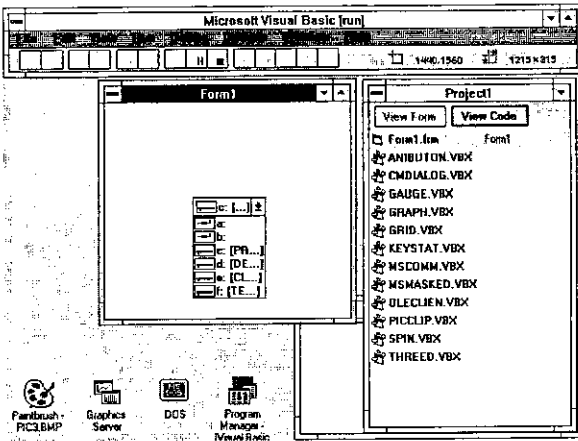
Finally, in the middle is the blank form called Form1. A form is a really a 'window' into which we put the controls and the dots are the snap-grid so that we can align the controls and resize them to match up perfectly. The granularity of the snap-grid is adjustable too. The form is the fundamental building block of

a VB program (although not compulsory) and one builds a VB program around a form and its embedded controls.

So to put the first control into the form we simply double-click on the button in the tool box that almost looks like a miniature disk drive. This puts a drop down list box into the form.



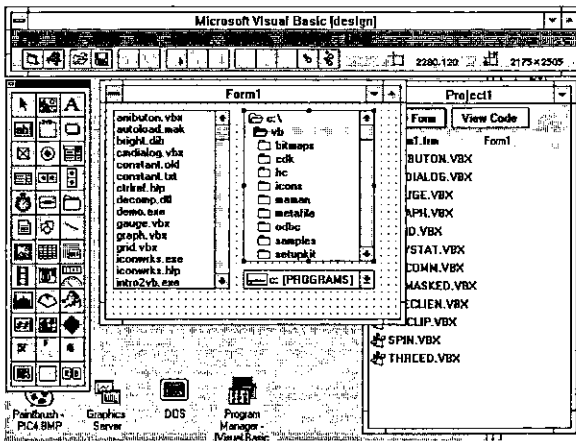
To see it work we just press F5.



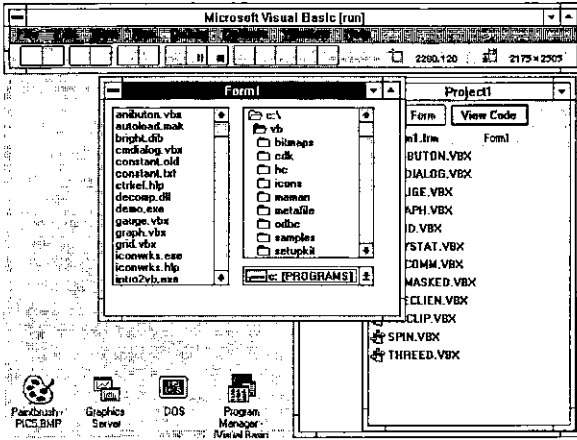
Voila! Form1 is now running and even now is a Windows compatible program. The picture shows the screen image after I have clicked on the drop down list to show my disk drives, so this program could become a little utility as it stands, and would be especially useful for networked PCs to show what disks are currently connected.

Now perhaps you are starting to see why I like Visual Basic — I have developed a little application with a resizable and movable window without writing a single line of Basic!

Let us move on. After stopping the program by double clicking on the little control box at the top left of the window, I added two more controls to Form1, a file list, and a directory list and I have moved and resized all of the controls into a pretty group. I have also made the window larger to cope with them all.

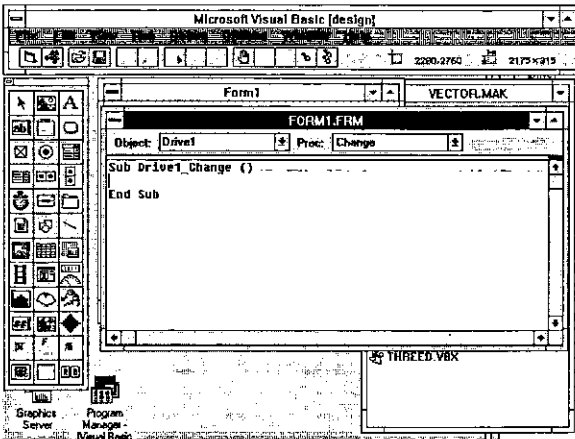


I press F5 to run this program and...

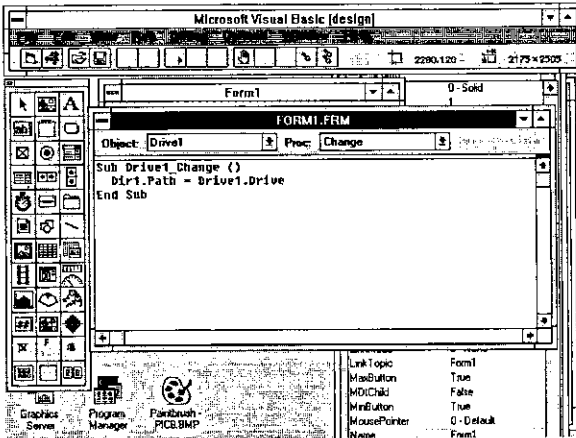


... the file list lists files, the directory list lists directories and the drop-down list lists disk drives. It all works with one proviso, namely that they are not connected. Changing a disk drive does not change the drive that the directory list is looking at, and changing the directory does not change the file list. So we stop the program and now we write some Basic.

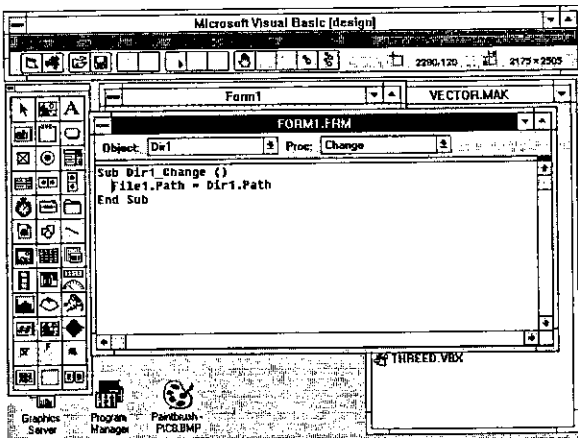
On the design screen I double-click on the drop down list of disk drives and get...



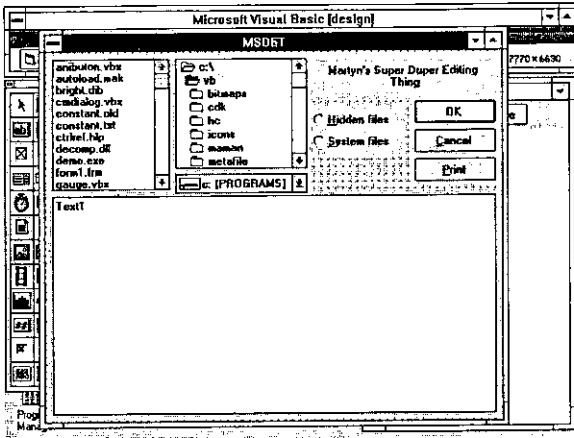
...a template for the Basic source code. When designing a form you can always jump to the Basic source code underneath a control by simply clicking on the control that activates it. I type in an assignment statement...



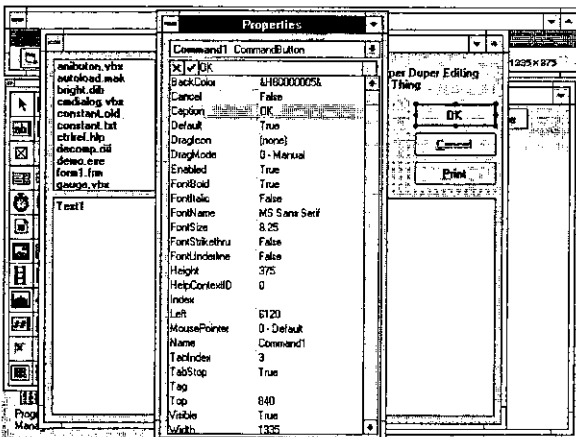
... which basically (forgive the pun) says 'Whenever Drive1 (the drop down list) changes, set Dir1 (the list of directories) to be at the new disk drive'. Then I save that and double click on the directory control (called Dir1) and type in some more code which says 'Whenever Dir1 (the directory list) changes, set File1 (the list of files) to be at the new directory'.



Now I have a fully working navigation structure which allows me to navigate all around my disk and select files at will — just like other real Windows programs which have File Open and Save As... options. I have only written two lines of Basic and all the rest is completely automatic. So now to build the full and final screen design...



...notice that I have managed to put in some 3D text, a couple of check boxes, some buttons etc. In order to change any of the attributes of, let's say, a button I can simply select it, press F4 and fill in the form.





All of the attributes on all of the controls can be filled in this way, and Visual Basic even allows you to change them when the program is running. This last feature means you could design a button which moves around the screen always running away from the mouse — never to be caught and depressed!

However, full Windows look and feel is properly supported. You can make forms and controls invisible, inactive, change states and anything else under program control simply by assignment statements. You can even assign to controls that are located on other forms. You can directly call the Windows API (which is documented) and call any available DLL routines if you have the appropriate documentation.

To round off this example I added a few extra lines of code to the buttons — two lines for the Cancel button, a few to the OK button, one to the print button, one line to each of the check boxes.

Windows has something called 'Accelerators' which means that if you press an Alt-key combination some action takes place. VB2 builds these in automatically, if you specify a button or menu option with a name like E&xit the & symbol underlines the x character which automatically becomes the ALT-X key accelerator. No coding necessary, but you can override this if you want.

To make the editor actually edit something, I added a few lines of code to the file list box which, when a file was selected, loaded a file and assigned it to the big text box (Text1). This is the edit area and it automatically creates a scroll bar when the contents become larger than height of the screen. Here is the loading code:-

```

Sub File1_Click ()
  ' comment: load a file for editing
  Dim Path$, record$
  ' comment: Get the file name
  Path$ = Dir1.Path
  If Right$(Path$, 1) = "\" Then
    Path$ = Path$ & File1.FileName
  Else
    Path$ = Path$ & "\" & File1.FileName
  End If
  ' comment: now load the file
  contents$ = ""
  Open Path$ For Input As #1
  Do While Not EOF(1)
    Line Input #1, record$
    contents$ = contents$ & record$ & Chr$(13) & Chr$(10)
  Loop
  Close #1
  ' comment: This one line updates the screen
  Text1.Text = contents$
End Sub

```

Some more Basic code is needed, but it is quite straightforward so I will not bore you with the detail. Adding a Windows style drop down menu bar across the top of the window is simply a matter of form filling again and then attaching code to each of the menu options. Even drag and drop is supported.

To improve this application I could make the edit area resizable when the window is resized (two or three lines of code), add Save File, Rename, Delete and Copy buttons to turn it into a little session manager.

## Colour and Help

There are a couple of other things I have not mentioned. The editor colour codes the source code by syntax, meaning that comments, keywords and variables are displayed in different colours. This greatly eases the understanding of the code. Finally, there is a built in context sensitive help. Double click on any keyword, press F1 and the help text describes it in detail. Do the same on any control and the help text describes the control in detail too. This means that if you want to learn Visual Basic you can simply buy the product, load it and press F1 for help — although some of the help text assumes you already know Basic.

## Summary

There is a lot more to VB2, I have only scratched the surface, but I hope that by now you can see that it is extremely simple to develop applications with it. If there could be a Visual APL I would be skipping on the desktops. Developers of this Visual APL would have to realise that screen presentation is very important, and the underlying language has almost become secondary to the screen image.

# Hooking up to the Internet

*by Dick Bowman*

*email:bowman@apl.demon.co.uk*

As we all know, the population density of APLers is not as high as we might wish; this makes us need to reach around the world to find the number of people we want to discuss APL topics with. Which is a problem, because long-distance phone calls cost money and the propagation delays of paper communications cost us immediacy.

But there is an answer, and I'd like to share with you what little I know so far.

I've addressed this to Vector, because the technology is only just becoming readily available to the individual in Britain now; the US has had better access for some time, and many companies already use similar facilities for their national and global communications.

The technology I'm talking about is Internet, and you can recognise someone who has access to it by their habit of sticking an email address with an '@' sign on it to everything they write. I'd been frustrated for some time because I found that people expected me to be on the Internet, but those who were on were exceptionally vague about how it all worked. So I found out, and this is what I want to share with you. None of the services, publications, etc. that I mention here should be construed as being recommended, praised or condemned; they just happen to be sources that I've used to achieve successful connection. By the nature of my geographical location my specifics will be specific to a person based in London, but the sources will point you to possibilities everywhere in the world.

## The Internet and Other Nets

The Internet is the world's largest computer network; but that said it's both anarchic and amorphous. It does not run as a commercial enterprise, so the first avenue for entry (look under 'I' in the phone book) doesn't work. The thousands of computers that comprise the net are mostly in educational and research establishments, and have hooked themselves together on a more-or-less voluntary basis. What you have to be able to do is to reach one of these machines.

Other networks such as CompuServe are not the Internet, although you can exchange email between these networks. Indeed I used CompuServe for this purpose for over a year. It's not perfect, but it works (within the documented limitations).

### Information Sources

Try asking around locally; you stand a good chance of success if you have connections into a local college or university. You might also write to RIPE NCC, Kruislaan 409, NL-1098 SJ Amsterdam, Netherlands (phone +31 20 592 5065); naturally they prefer email (ncc@ripe.net). There's also a book packed with useful information (remember, this anarchy does not provide a user manual) called 'The Whole Internet Users Guide and Catalog', written by Ed Krol, published by O'Reilly and Associates Inc. (ISBN 1-56592-025-2).

### Types of Connection

From now onward I'm going to assume that you've decided to pursue this with the aim of getting your PC connected, because this is all I have personal experience of. The Internet is, to some degree, oriented towards UNIX but it truly is an agglomeration of all sorts of hardware and software. Suitable software for PCs, Amigas, Macs, UNIX and so forth is all quite readily available (and a lot easier to get once you're connected — this is infuriating unless you feel well motivated at times).

There are two choices to make; either be a user at an established site, or become a site of your own. So if you call up your local college and they say OK you end up as mrapl@uhull.edu.uk (you choose the bit that goes before the @ in your email address). I did it the other way (which lets you choose some of the bit that comes after the @).

### Who Pays for All of This?

I just knew you'd ask sooner or later; the glib answer is 'the US Government, of course'. Yes indeed, ole Mom and Pop in their Kansas grocery store are paying to let you and me exchange email with our pals in the Obrinsk nuclear research facility. It's not quite as ideal as that, but every site covers its own cost as an overhead — you don't pay Internet Inc. (how could you, they don't exist).

The corollary to this is that there is a policy of 'acceptable use'; what you do on the Internet ought to be connected in some sense with education or research. The policy isn't rigidly monitored, but it would be bad form to take advantage of

Internet connections to operate a company email network between your London and New York offices.

What you'll have to pay is the hardware and software cost of setup, whatever your service provider charges, and telephone costs (to reach the service provider, not the same thing as the services you use).

### **The Financial Good and Bad News**

I connect to a service provider called Demon Internet Services, who are based in London; their charges are £12.50 for initial enrolment and a monthly charge of £10. No usage charges. This is possibly a little more than I used to pay for CompuServe — but all I did there was to send and receive Internet mail.

The financial down side is that your service provider will probably encourage you to use at least a V32 modem (9600 bps); this is not unreasonable from their perspective because they've got a finite number of ports and if all of you folks with your 'by appointment to Queen Mary' mahogany-cased 300 baud acoustic couplers hog the lines all day then the service provider is going to have to buy some more lines and modems (and pass the cost on). You'll probably find that it's good news for you as well, because the modern-day email user has lost the knack for economical mail that always made the I P Sharp system such a pleasure to use.

The financial up side is that you will not need to splash out on a PC comms package, because you'll get a special one from your service provider. This is because your machine is becoming a full-fledged (if leaf-only dialup) node on the Internet, and this runs on TCP/IP — you're going to have a whole lot more going on than type a few keys, press Enter, wait for response.

### **The Software**

Once again the hardened Internet nutter is going to tell you to download the software. Great, you don't have it yet and you're not paying for that fast modem until you know the service is worth having. Never mind, they'll probably sober up long enough to send it to you on diskette.

Now, about this software...

It's awe-inspiring stuff.

Take a backup. Take two, they might come in handy.

What I have is three things called KA9Q, SNEWS and PCELM; they all conspire to work together in one directory — but there are a few loose ends lying around and we've ended up with a few new environment variables. Along the way I encountered a totally trashed hard drive, DOS software that insisted on directory names being separated by forward slashes, DOS commands that produced gibberish, a CMOS clock that totally lost track of time and a PC that decided Eastern Standard Time was the timezone to be in.

But it all (sort of) works now, and on a scale of 1 to 10 I rate this stuff as harder than APL, but simpler than Visual Basic. The user interface is rudimentary and I'm not convinced that the file structures are robust enough to survive any sort of software failure. Nevertheless I can do the three major things that I want Internet access for — and who knows, there may be better stuff for me to download (or I might write myself some tools to do some of the manipulations I need).

## Email

My principal motivation was to be able to send and receive email. Easy to do — just fire up PCELM and write the messages (handy tip — the PCELM configuration file has a parameter called 'edit' that lets you call up a simple text file editor, like DOS's 'edit', a configuration line that says 'edit edit' is not the best idea you'll have all year). Next time you start up KA9Q all your outgoing mail gets sent and you receive all that's coming your way. This may include mail you sent earlier being bounced back because of bad addresses — although if this happens it often happens while you're logged on to the network, surprising. Go back to PCELM to read and reply to the mail.

I'd better warn you of two types of characters you'll encounter sooner or later. There's the Email Vacuum Pump, who thinks that reading email is an adequate activity and never replies to anything. Then there's The Phantom, who sends you mail that you can't reply to because the address they gave you is bad according to the system. Both of these characters are more numerous on the Internet than they are on other systems.

## News

News is a sort of mutated mail, that gathers together messages on related topics and sends you the whole lot. You can elect to join news groups, receiving all the traffic and making some of your own if you like. For example, if you have an interest in zymurgy, then there's a news group. If you have less common interests, then you can join a newsgroup called 'comp.lang.apl'.

News also makes you realise why that fast modem is important; my first news download got me more than 600 messages — all from my local service providers newsgroups. By the time a mail message has been through the mill a few times with bits appended, and with the Internet routing stuff glued to the front it can be quite hefty (none of the IPSA To: BOW From: BOW 'No' succinctness here).

You elect to join (and leave) mailgroups through all of the software you got as a starter set — mostly it's SNEWS, but there's other stuff scattered around as well. Incidentally, you may not get a backlog of news as soon as you join a newsgroup; it took a day or so before comp.lang.apl started to trickle in.

Once your machine has grabbed the news (happens automatically when you dial in) you have to run a local program called UNBATCH to distribute it in a sort of orderly way into the newsgroups you set yourself up with before reading it with SNEWS. As I write this most of mine goes into place called 'junk' because I'm being fed more than I've organised myself to deal with — but I can still read it there. I get around about 50 news items per day and anticipate this figure remaining fairly constant as I disconnect from the irrelevant and hook myself into the more personally interesting. Thus proving that fast modems do not save money on phone bills — they just let you grab more stuff. A program called EXPIRE lets you clear off the old stuff, but I haven't found a way yet to build an archive of the interesting ones only, maybe I have to do that myself.

There are literally hundreds of news groups; I downloaded a 500k byte file giving details — it almost certainly is not all that are available.

## FTP

File transfer is a real gem; if someone on the Internet with a machine that allows logins wants to make a file available to other users then they can do it. You don't even need an account on the machine you're grabbing the file from. The facility is called 'anonymous ftp', so if you know what you're looking for and the name of the machine all you have to do is start up ftp and log on as 'anonymous'. Even if you don't know precisely what you're looking for but you know a place to look you can still log on and move around the directories.

Naturally, there are limits to all of this, and the information providers choose what to put in the accessible places. But if the file's there, and you're allowed to reach it, all you have to do is get it and it's on your own machine. Remember, this is education and research, no money changes hands; you're dealing mostly with a country that has open access laws for things paid for out of the public purse.

## Other Resources

Krol's book talks at length about Archie, Gopher, WAIS and the World-wide Web; all ways of helping you find information you need on computers connected to the Internet. Not all of these are available on all hosts, and I really haven't had time to explore any of it. You may be interested to know that even though Internet really isn't about APL at all one of the resources singled out by Krol is WAIS. Much of the development for WAIS was done at Thinking Machines — remember them?

Telnet may or may not prove useful; it lets you log on to other computers, wherever they may be, for your local phone call. Not everyone will let you have an account, but some will. Your activity may not be regarded as benign if you use a computer in Japan just for the hell of it. This may rebound on the other people who are benefitting from the appropriate use of a very significant resource.

## Summary

This is being written while most of this is very new to me; I would hope that by the time you read it I'll know much more. I've used email as a way of keeping in contact with APLers around the world for several years; it works very well. I thought it was important to write this at an early stage in my direct connection to Internet because if I left it any longer I'd join the ranks of 'oh sure, just sign up'.

Other networks exist, and their specific natures may suit your purposes very well. Nothing except money makes this an either-or situation. If you're on another network (like CompuServe) we can still exchange email; we can share more if you're on Internet directly (and to be fair, you can share things with other CompuServe accounts that you can't let out to us).

Give it a try, you have only your hard disk to lose. The more of us there are reading `comp.lang.apl` the more of us there are going to be writing to `comp.lang.apl`, and the higher the chances of finding really useful stuff every day.

Can't get the software going? Send me an email.



---

# 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

The Challenge of the New	Duncan Pearson	120
Technical Correspondence		122
Sharing the Spoils or Circling the Square	Mike Day	123
An Exchange on Primes	Roger K.W. Hui	130
Span Representation:		
Improving the Display of J Verbs	Richard Oates	135
An APL Truetype Font for Vector	Adrian Smith	138

---

# The Challenge of the New Object Programming and the Windows GUI

by *Duncan Pearson*

Until about two years ago there was fairly common agreement on what was good practice in the coding of APL systems. What would come out was generally a system navigated by a simple menu function which would call at each stage a number of small black-box functions that would do fairly simple specific things to their arguments and return the results. What you were doing (and what you were doing it to) would be fairly well defined by your position within a hierarchy of functions at the bottom of which would be *GO* or *ΔLX*.

The interaction with the user was very well controlled. He could choose an option from a menu or edit some data (usually a small subsection of the whole). Even with this small choice we would quite often end up with a few large functions in which we would interrogate the user then branch to one of say ten labels depending on his choice. Having done what he asked we might return him to the menu and ask him what he wanted to do next. This would continue until he chose to leave that part of the application. At each stage the option that he chose would probably act on data local to that part of the system. So far so good.

Now imagine a situation in which the user can choose one of two hundred different actions. As I sit in Word for Windows I have at least that many from which to choose. What is more I spend over 90% of my time in the base state — editing the document. How do you manage it? Do you have great wadges of global data which is all manipulated by two hundred different little functions, and have a two-hundred-and-three-line function which loops round deciding which of these little functions to call? No, what you do is to associate with everything the user can do a 'callback' function which will automatically be called whenever that action (such as pressing a button or selecting an item on a menu) occurs.

This is the standard Windows approach to the problem and it gets rid of your 203-line function. However, as you never know which of these functions will be called next, or what condition the system will be in when it is called, you must do something to ensure that any data it needs is readily available. The easy but dangerous approach to this (adopted by Manugistics for their APLGUI — see the review in this issue) is to leave everything lying around globally in the workspace. This leaves you with a new problem. How do you, as a developer

starting to maintain someone else's broken system, know what the hell does what, with what, when?

One well trumpeted approach to this problem is to use data objects and pass messages to them to update them. I can well believe that if a system is designed from the bottom up using this approach then the problems of maintenance would be dealt with. One of the difficulties of the use of object methods within the Windows GUI offered by the current APL systems is that the object paradigm extends only as far as the management of the GUI. The main area where benefits are to be reaped is in being able to define a new data object class and then to generate instances of that object class as and when they are required. Not only is the object paradigm restricted to the GUI but only the standard supplied graphics objects are true object classes. In Dyalog APL the 'button' is a standard class of graphics object of which it is simple to define instances, but if for example I want a class of specialised buttons that change the cursor when it crosses them, then I need to define a special function with which to generate instances of that class.

The basic GUI is the obvious place for object programming to start, but it leaves the programmer a nasty choice of either generating a number of hybrid systems using only the GUI object classes defined in the interpreter, or of implementing a complete object definition and management system in APL. Why, one might ask, not just program in Smalltalk and be done with it. Roger Hui made a telling comment in his talk to the BAA in February: when asked if he found all those funny letter combinations in J to be a hindrance to programming he said that it wasn't the language that you used but the way that you thought — one in the eye for the 'symbols are everything' brigade. If we cannot program for the new user environment effectively without having to create an artificial 'programming environment' layer then perhaps we are in the wrong language.

Which would take more work and give a more durable and flexible result, putting the programming environment of Smalltalk into APL or putting the array capabilities of APL into Smalltalk?

# TECHNICAL CORRESPONDENCE

## More on APL Packages

From: A.J. Brown

1 February 1993

I read with interest the article from Nicholas Small in Vector 9.3 (page 100).

The same problem had occurred to me some time ago, using APL\*PLUS II. My solution was to write the attached utility. The reasons for the name are now lost in the mists of time, but this function is now an integral part of all my workspaces.

```

v Δex;ΔΔΔA;Z
[1]  ⍎ 921124
[2]  ⍋WINDOW← 0 0 25 80 ⍋TCFF
[3]  ⍎ Delete any programs not needed
[4]  ⍋EX 'ΔpΔ Δunpost Δfind h PROG DISPLAY PACKVR UNPACKVR ORDLOC
      RELABEL STORAGE'
[5]  ⍎ ΔΔex - matrix containing list of variables NOT to be deleted
      (must contain ΔΔex)
[6]  Z←⍋EX(~/ΔΔΔΔA^.=⊆ΔΔex)/ΔΔΔA+{(1↑ρΔΔΔA),~1↑ρΔΔex)+ΔΔΔA+⍋NL 2
[7]  ⍋FONTE ⍋FNUMS
[8]  0 ⍋INBUF 'SIC'
[9]  ~1 ⍋INBUF 13
[10] ~1 ⍋INBUF 'SAVE'
[11] ~1 ⍋INBUF 13
[12] ~1 ⍋INBUF 'START'
v

```

A.J. Brown  
 Cygnus Enterprises Limited  
 15 Gardenfields  
 Stebbing  
 Dunmow  
 Essex CM3 3RG

# Sharing the Spoils or Circling the Square

by Mike Day

Just before Christmas last year, I was having lunch with my friend Mike in his Government department's canteen. We studied Maths together many years ago and are still "Scientific" civil servants. He wondered if I could help him solve his rounding problem. His unit determines the amount of grant payable to a number (say 50) of "clients" under a few (around 4) different accounting headings. However most clients bid for less than all 4. His ministry has formulae to allocate the available funds resulting in integral amounts of pounds (let us say) to each bidder for each heading. However, they only pay out half of the approved funds.

Mike had the task of making the tables of Ministry expenditure look right in whole numbers even though halved from the original approved levels; the sum of the rounded halves must be as near as possible the same as the rounded half of the sum for any client or for any heading or indeed overall.

By the way, Mike has Excel for Windows but not APL. He has always taken a guarded and rather bemused view of my interest in it.

We can restate his problem only slightly more formally as:

## Problem 1

Seek a way of arbitrarily rounding each half-integral number in a matrix up or down so that the absolute cumulative error in each row, each column, and over the whole array shall be nowhere greater than one half unit.

As a smaller example than the real thing, consider 5 clients and 3 headings with these agreed (100%) spends:

Figure 1

	1	2	3
1	33	0	55
2	0	120	30
3	0	67	0
4	23	29	43
5	0	0	37

We only need worry about the odd cells as the even ones stay integral correctly. Mark the odd cells in a Boolean matrix:

Figure 2

	1	2	3
1	1	0	1
2	0	0	0
3	0	1	0
4	1	1	1
5	0	0	1

Mike's problem is effectively to replace each one in figure 2 by a plus or minus one so that all the marginal sums are 0 or +1 or -1. One solution is easily seen to be:

Figure 3

	1	2	3	
1	1	0	-1	0
2	0	0	0	0
3	0	-1	0	-1
4	-1	1	1	1
5	0	0	-1	-1
	0	0	-1	-1

I've appended the totals. So the complete result for the rounded halved matrix is:

Figure 4

	1	2	3	
1	17	0	27	44
2	0	60	15	75
3	0	33	0	33
4	11	15	22	48
5	0	0	18	18
	28	108	82	218

Compare the unrounded version:

Figure 5

	1	2	3	
1	16.5		27.5	44
2		60	15	75
3		33.5		33.5
4	11.5	14.5	21.5	47.5
5			18.5	18.5
	28	108	82.5	218.5

Mike has solved his real problem by inspection but he thought there must be an algorithm to do the trick. I agreed but at first I couldn't see how. Then I saw the light. Look at Figure 3 again — but change the signs of all the margins except the total:

Figure 6

	1	2	3	
1	1	0	-1	0
2	0	0	0	0
3	0	-1	0	1
4	-1	1	1	-1
5	0	0	-1	1
	0	0	1	-1

The marginal totals of the 6 x 4 table are all zero! We have an extended problem which is simpler than the original:

### Problem 2

Given a Boolean matrix with even numbers of ones in each row and column, replace each one by plus or minus one so that all the marginal sums are zero. The (- 1 1) DROP of the solution is a solution to PROBLEM 1 and the negation of each marginal total except the overall cell is the sum for the corresponding reduced row or column.

How do we set up the extended Problem? It is easy enough in APL to adapt our matrix: append a new column which is the 2's modulus of the sum of each row. Do the same with the columns and append a new row. So, if *M* is the original matrix,

$$ME \leftarrow 2 \mid x, [1] \neq x \leftarrow (ME, +/ME)$$

It's neater to defer the parity check. We still need to solve the extended table, *ME*! Consider this problem:

$$M \leftarrow 1 \ 1 \ \rho \ 1$$

Figure 7

which is extended to

$$ME$$

Figure 8

$$\begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix}$$

trivially solved by

$$\begin{matrix} + & - \\ - & + \end{matrix}$$

Figure 9

The solution can be traced in a cycle from 1 to 4 (to 1) with alternating signs:

$$\begin{matrix} 1 & 2 \\ 4 & 3 \end{matrix}$$

Figure 10

The next most trivial problem is the 2 x 2 matrix:

$$\begin{matrix} + & M & + & 2 & 2 & p & 1 & 0 & 0 & 1 \\ 1 & 0 & & & & & & & & \\ 0 & 1 & & & & & & & & \end{matrix}$$

Figure 11

extending to

$$\begin{matrix} & ME & & & \\ 1 & 0 & 1 & & \\ 0 & 1 & 0 & & \\ 1 & 0 & 1 & & \end{matrix}$$

Figure 12

We see that there is still an easy solution:

$$\begin{matrix} + & & - \\ - & - & + \\ - & + & \end{matrix}$$

Figure 13

This again equates to the assignation of alternating signs to a cycle of cells in the order 1 to 6:

$$\begin{matrix} 1 & & 2 \\ . & 4 & 3 \\ 6 & 5 & \end{matrix}$$

Figure 14

Why does it work? In establishing an even parity of ones in rows and columns in the matrix, we allow the existence of complete rectangles of 4 points: any candidate (e.g. 2) must have 2 mates, (1 and 3). There is evidently a missing point opposite 2. However it closes the rectangle whose other corners are 4 5 and 6. We can therefore regard it as doubleton of plus and minus ones cancelling out to zero.

Of course sometimes, as in figure 8, the rectangle is complete and does not need completing with a virtual point.



And that's it! Start with a plus at an arbitrary 1 in the extended Boolean matrix, and move in a city grid path, i.e. alternately across (left/right) with a plus and up/down with a minus stepping through remaining target cells in the array. I actually set all the ones to value two to start with so that I can use the same array throughout rather than have one for Booleans and one for the new values. As real problems tend to have more than one closed path, we need to be able to choose a new arbitrary starting point from time to time.

Here's a bigger example:

```
(4 5ρ' '),SHOW BIGM
+++ ++ + +++ ++          + + + ++ +   ++ ++ +
+ + + + + ++ +++++ + ++ + ++++++ +++ ++ ++++++ ++
+   + +++ +++++ +++++ +++ +++++ + + ++ +++ +
++ ++++++ +++++ + + +++++ + ++ +++++ +++++   + +

(4 5ρ' '),SHOW SOLVE BIGM
++- -- + -++ -+         - + - -+ +   -+ -+ -
+ - - + + - -- +--+ - +- + +--+ -+ - -+ -+--+ -+
-   + --+ +--+ -+--+ -+--+ + - -+ -+--+ +
-+ +--+--+ -+--+ - + -+--+ + -- +--+--+ +--+   + -
```

I've set SHOW's  $\diamond$  characters to blank for readability on the printed page.

It is all fairly simple and I hardly need bother you with the simple code. It's in the appendix. I used my own version of Direct Definition under APL\*PLUS/PC 8 to set up utility functions with somewhat meaningful names. I hoped in this way to keep the main function *SOLVE* quite undaunting for my non-APLer colleague, looking deliberately quite FORTRAN-ish! This did indeed allow Mike to learn enough Excel macro/programming language to mimic the APL functions in around 100 lines.

It would of course have been a good opportunity to use Dynamic Data Exchange with a Windows-based APL as described by Adrian Smith in Vector 9.2, 1992. Adrian has drawn my attention to a Philip Benkhart's paper "A Dance of Rounds" (APL Quote Quad 1991). He deals with the more general problem of distributed rounding of arrays of fractional numbers rather than strictly half integral ones, and considers relative errors as well as absolute ones. As I do not have a fully implemented second generation APL yet, I am not sure whether his approach would solve Mike's poser. I suspect it would!

I haven't yet succeeded in generalising my algorithm to 3 or more dimensions. "Why?" said Mike. "What about more than one year?" said I. In investigating the higher dimensional question I did derive a rather neater set of functions than those you see: there's only one *INDEXNEXT* function which handles each

dimension in turn. In three dimensions you go across, down, back. Everything is done on the ravel of the Boolean with a tricky one-liner to return the indices of the required one-space which is then checked for the next candidate. Unfortunately it only works in one and two dimensions. So I hope Mike's users don't add up across years!

I suspect that one way to generalise to n-dimensions is to develop my explanation of why the 2-d method works: each candidate point must have  $n$  neighbours in the extended array. That leaves  $(2*n) - (n+1)$  possibly virtual points to complete the corners of the n-rectangle. In order to catch Vector's deadline I leave it as an exercise!

## Appendix: Function Listings

```

      A N.B. []IO+1 throughout
      ΔΔL ΔL
ADCOMMENT: ω A.. COVER/UTILITY FNS FOR MAIN FN 'SOLVE' ...
COL: ω[;α[2]] A      find column α[2] of matrix ω
ROW: ω[α[1];] A     find row α[2] of matrix ω
FIRST:1+ ω A        return first element of a list
LAST:-1+ ω A        return last element of a list

INDEXNEXT: ω MINDEX WHERE ω
      A find the indices of the next untreated cell

INDEXNEXTINCOL:x:NONE x+(WHERE α COL ω ),LAST α :INDEXNEXT ω
      A next in ω[;[α[2]]

INDEXNEXTINROW:(FIRST α ),WHERE α ROW ω
      A find next cell in ω[α[1];]

MINDEX:,1+(ρ α )τ-1+ ω
      A index pair for α-dim matrix given ravelled index ω

WHERE:ι0:2ε ω +, ω :, ω ι2
      A find the next untreated cell in a vector

NONE:2>ρ ω A        check if at least two elements in list

SHOW:'-o+'[2+ ω ]
      A display a matrix of +1 -1 or 0 as + - or o

SOLVER:SHOW y,[ι]+fy+x,+/x+SOLVE []+2>? ω ρ α
      A do a problem size ω, sparsity +α

ODD:2| ω A mark all odd elements

```

*EXTEND: 2×ODD w , [1]+/ w + w , +/ w*  
*a append rows + cols, check parity, double*

*TRIM: -1 -1 + w a remove appended row and column*

```

v SOLVE[[]]v
[0] z+SOLVE m;i;[]IO
[1] a Solves half integer matrix rounding problem
[2] a by following city-grid route in an extended even parity matrix.
[3]
[4] a The style is deliberately FORTRAN with most 'APL' consigned to
[5] a direct definition one-line utility functions
[6]
[7] a boolean matrix m
[8] a z has +/- ones in place of m's ones.
[9] a Magnitude of all row + col + total sums ≤ 1
[10]
[11] []IO+1
[12] m+EXTEND m a append row+col, adjust parity, double
[13] i+pm a arbitrary seed
[14] loop:
[15] -(NONE i+1 INDEXNEXTINCOL m)/end a next cell, same col if poss,
[16] a finish if none anywhere
[17] m[i[1];i[2]]←1 a set cell
[18] i+1 INDEXNEXTINROW m a next cell in row. MUST be one!
[19] m[i[1];i[2]]←-1 a set it
[20] →loop a repeat until done
[21]
[22] end:
[23] z+TRIM m a clean up the answer
v

```

# An Exchange on Primes

by Roger K.W. Hui

USENET is a world-wide network connecting approximately 37,000 machines in academia, government, business, and other organizations (see reference [1]). It has a "news" subsystem of discussion groups, including the APL group "comp.lang.apl". Over the last 20 months, the group averaged about 2.3 messages per day. The following exchange took place recently:

```
From: mitloehn@exaib.wu-wien.ac.at (Johann Mitloehner)
Subject: explicit <-> tacit, space/time usage on prime numbers
Date: Fri, 30 Oct 1992 10:55:15 GMT
Organization: Wirtschaftsuniversitaet Wien
```

Here are some experiments (by T. Kolarik and myself) with the well-known primes-idiom on several machines, comparing tacit and explicit definition of the verb:

On an HP 9000/720 (HP-UX):

```
eprimes =. '(i.y.)#~2=+/0=|/~i.y.:' NB. explicit def
tprimes =. (=2 @ (+/ @ (=20 @ |/~ @ i.))) # i. NB. tacit def
time =. 6!:2
space =. 7!:2
ts =. (time, space)
ts 'tprimes 100'
2.71 2528
ts 'eprimes 100'
0.15 464
```

[Similar timings on a Mac Quadra and Mac IIX.]

Result: explicit definition is much faster.

I have done a similar thing on a Mac Classic with J 5.1a; interestingly, tacit was slightly faster (the difference being very small; I did no space measure).

But tacitly defined verbs are supposed to be (much?) faster, aren't they? Or is this just a bad example? (Or is our tacit version so bad):

```
-----
From: sam@csi.jpl.nasa.gov (Sam Sirlin)
Subject: Re: explicit ' tacit, space/time usage on prime numbers
Organization: Jet Propulsion Laboratory, Pasadena, CA
Date: Wed, 4 Nov 1992 19:33:52 GMT
      [Quotes from Mitloehner's message deleted.]
```

```
Using J6 on a sun sparystation 1:
  ts 'tprimes 100'
4.53315 24
  ts 'eprimes 100'
0.099996 24
  ts 'eprimes 100'
0.116662 24
  teprimes=. '(i.y.)#~2=+/0=|/~i.y.' : 11 NB. tacitize eprimes
  ts 'teprimes 100'
0.116662 24
So not all tacit definitions are equal.
```

```
-----
From: eem@ipsaint.ipsa.reuter.COM (McDonnell, Eugene E.)
Date: Sat, 7 Nov 92 22:00:03 UT
s=. '(i.y.)#~2=+/0=|/~i.y.'
eprimes =. s : ' NB. explicit def
tprimes =. (=2 @ (+ @ (=2 @ |/~ @ i.))) # i. NB. tacit def
cprimes =. s : 20 NB. :20 replaces obsolescent :11
lr=.5!:5 [linear representation]
lr <'cprimes'
i.@] #~ 2&=@(+@ (0&=@(|/~@ (i.@))))
time =. 6!:2
time 'eprimes 100'
0.116667
time 'tprimes 100'
3.53333
time 'cprimes 100'
0.1
```

Herr Mitlöhner is correct: tprimes is not equivalent to eprimes.

Using the :20 compiler on s gives a tacit form cprimes whose logic is that of eprimes, and which is more efficient than the explicit form, by a small amount.

These measurements were made on a Mac Quadra 700.

```
-----
From: hui@yrloc.ipsa.reuter.COM (Roger Hui)
Subject: Re: explicit ' tacit, space/time usage on prime numbers
Organization: Iverson Software Inc.
Date: Sat, 7 Nov 92 23:38:13 GMT
      [Quotes from Mitloehner's message deleted.]
```

In July 1990, at the APL90 Conference in Copenhagen, Eugene McDonnell presented timings indicating that a tacitly defined nub was faster than an explicitly defined nub, and was as fast as the primitive ~ ..

I was in the audience and was surprised by this result. Tacit defns arise from the use of adverbs and conjunctions, function assignment, and forks; that they happen to be faster is an unexpected benefit.

The slower times for tprimes obtained above can be accounted for by the difference between =&0@|/ vs. =&0@(|/). I applied the tacit translator to the explicit defn, and obtained the following time/space results (J6 on a PC486/33):

```
ts 'tprimes 100'
2.86 24692
ts 'eprimes 100'
0.05 53420
f0 =. '(i.y.)#~2=+/0=|/~i.y.': 11
ts 'f0 100'
0.05 50312
lr =. 5!:5
lr <'f0'
i.@] #~ 2&=@(+/@(0&=@(|/~@i.@]))))
```

This shows that tacit defn is at least no slower than explicit defn. The following table shows that it is in fact faster. All times are obtained using 100 time 'f n' on J6 PC486/33.

	200	100	10	5	1
tprimes	11.0334	2.8456	0.0341	0.0109	0.0033
eprimes	0.2230	0.0582	0.0049	0.0044	0.0044
f0	0.2213	0.0565	0.0032	0.0027	0.0027
f1	0.2208	0.0566	0.0033	0.0027	0.0026
f2	0.2208	0.0566	0.0033	0.0027	0.0027
f3	0.0193	0.0138	0.0038	0.0033	0.0038
f4	0.0143	0.0088	0.0017	0.0016	0.0016

f0 is faster than eprime by a fixed amount, 0.0017. This is consistent with the explanation of why tacit is faster than explicit: a tacit defn is not reparsed on execution, and 0.0017 is the time it takes to parse eprimes. The relative advantage of tacit over explicit is thus greatest for small vectors, when parsing time dominates, and diminishes as vector size increases. This fact is of interest to compiler writers.

f1 is an attempt to see how much better or worse a "hand-written" tacit defn may be. f0=.1.@] #~ 2&=@(+/@(0&=@(|/~@i.@])))), the defn produced by the translator, is periphrastic (*from the Greek "periphrasis": a round about expression. Ed*). It is easy to see that the two uses of @] are unnecessary, and the use of passive can be eliminated by rearranging the tines of the fork. As well, I tend to shy away from deeply right-nested compositions. Putting it all together:

```
f1 =. 2&=@(+/@(0&=@(|/~)@i. # i.
```

The timings in the table indicate that *f1* is slightly faster than *f0*. Not too surprising, as *f1* is only slightly different from *f0*.

As indicated at the start, speed is not the main point of tacit defn. Some of the advantages of tacit defn were outlined in a msg two weeks ago:

- 0) The tacit form encourages the use of components, primitive and user-defined. Use of components is one of the few effective ways of dealing with complexity.
- 1) The tacit form focuses on how components are combined, i.e. it focuses on the interface between components, where errors tend to occur.
- 2) Tacit expressions are algebraic, i.e. they are amenable to proof and other formal manipulations.
- 3) Tacit expressions are constructed using operators (adverbs and conjunctions), ideas debugged in mathematics over hundreds of years.

Even for something as small as this primes problem, it is possible to see how these benefits may be applicable. *f1* can be redefined to use a sub-function, of interest in its own right:

```
sieve =. 2&&@(+/@)(0&&=)@(|/~)@i.
f2    =. sieve # i.
```

*f2* is slightly clearer than the others, in bringing into sharper relief the structure of the computation. It is about as fast as the others (see table).

I'd previously written a recursive version of primes which avoids generating the  $n^2$  table:

```
t=.i. 0 0
t=.t,'$.=.1+30<y.'
t=.t,'y.>: # ]]2 3 5 7 11 13 17 19 23 29'
t=.t,'p,(*/0~:p|/s)#s=.k+i.>:y.-k=.{:p=.f3 <.%:y.'
f3 =. t :''
```

A hand-translated tacit version of same:

```
basis =. (>: # ]]&2 3 5 7 11 13 17 19 23 29
sieve =. */.@(0&&~:)@(|/)
suffix =. (:@[ ([ + i.@-.-) ]
extend =. [ , [ (sieve # ])] suffix
f4     =. ($:@<.%: extend)]'basis @. (30&>)
```

The time and space complexity of *f3* and *f4* are of a lesser order than the others, so that the advantage shown in the table increases with argument size:

```

100 time 'f0 500'          100 time 'f0 1000'
1.3583                    ws full
100 time 'f3 500'          100 time 'f3 1000'
0.038                      0.09
100 time 'f4 500'          100 time 'f4 1000'
0.0335                     0.0829

```

Finally, 2) speaks of tacit expressions as being more amenable to proof and other formal manipulations. Optimization is a class of formal manipulation, and in the future the implementation may exploit formal properties of tacit defs to effect faster execution. I described some possible approaches at the Minnowbrook '92 conference in mid September.

Tacit definition is discussed in an APL91 paper [2]. Briefly, an explicit definition is one which explicitly mentions its argument, as in `sum=.'+/y.'`; a tacit definition is one which does not, as in `sum=.'+/`. The tacit translator mentioned in the messages is also discussed in [2].

The "well-known primes idiom" in the original message computes the  $n$  by  $n$  residue table of the integers from 0 to  $n-1$ , sums a mask marking the divisors, and selects the integers with exactly two divisors. In contrast, the recursive algorithm in `f3` and `f4` computes a vector  $p$  of the primes up to the square root of  $n$ , then computes the  $(\#p)$  by  $n-k$  residue table of  $p$  against the integers from  $k$  to  $n$ , where  $k$  is the last element of  $p$ , and returns  $p$  suffixed by integers from  $k$  to  $n$  having no divisors in  $p$ . The following dialog illustrates the internal workings of the algorithm:

```

<.:%: 40          f4 6          basis 6
6          2 3 5          2 3 5
(f4 6) suffix 40
5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
          28 29 30 31 32 33 34 35 36 37 38 39 40
(f4 6) sieve (f4 6) suffix 40
0 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0
          1 0 0 0
(f4 6) (sieve # ]) (f4 6) suffix 40
7 11 13 17 19 23 29 31 37
(f4 6) extend 40
2 3 5 7 11 13 17 19 23 29 31 37
f4 40
2 3 5 7 11 13 17 19 23 29 31 37

```

## References

- [1] Cerf, V.G. *Networks*, Scientific American — Special Issue on Communications, Computers, and Networks, 1991 9.
- [2] Hui, R.K.W., K.E. Iverson, and E.E. McDonnell, *Tacit Definition*, APL91 Conference Proceedings, 1991 8.



# Span Representation: Improving the J Display of Verbs

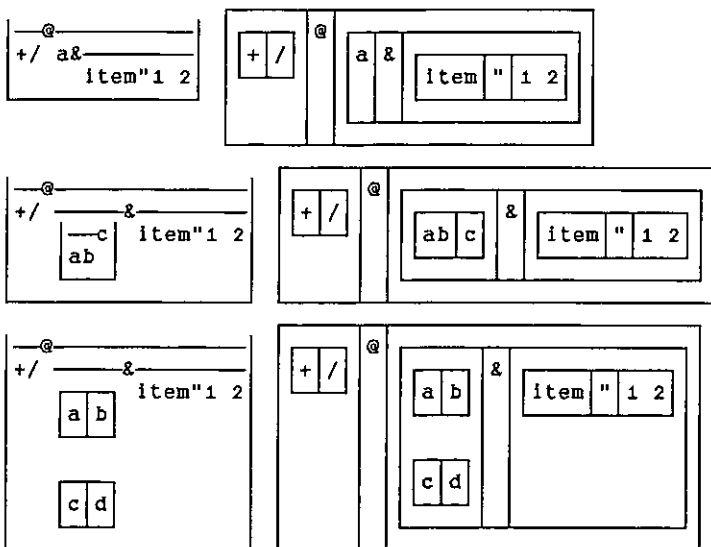
by Richard Oates

## Abstract

Verbs appear in a figure which is conceived for nouns. It can describe any rank but the representation of a verb is a list. Verbs absorb more thought than nouns and the population is bigger. Can the spacious noun figure be deflated for the verb?

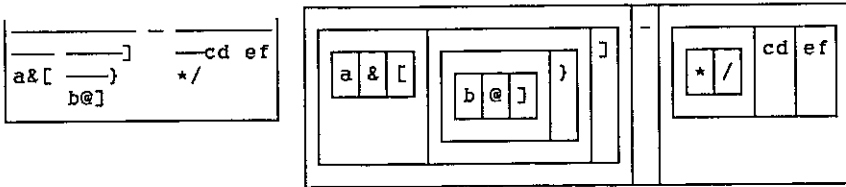
## An Initial Simplification

Omit the inner bottoms and sides, and also the tops that do not unify (span) some part of the sentence. Swap the outer bottom, sides, and top for a cup to conceal the inner structure.

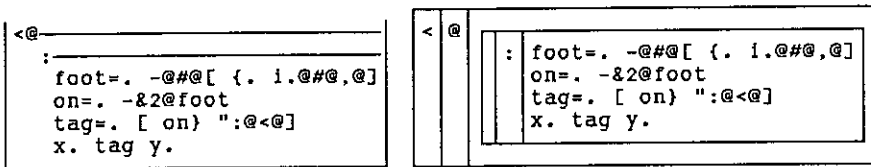


## Removal of Excess Ink

Retain and make blank the side of a box which appears between two spans, or adjacent to a Minus primitive, or between two names.

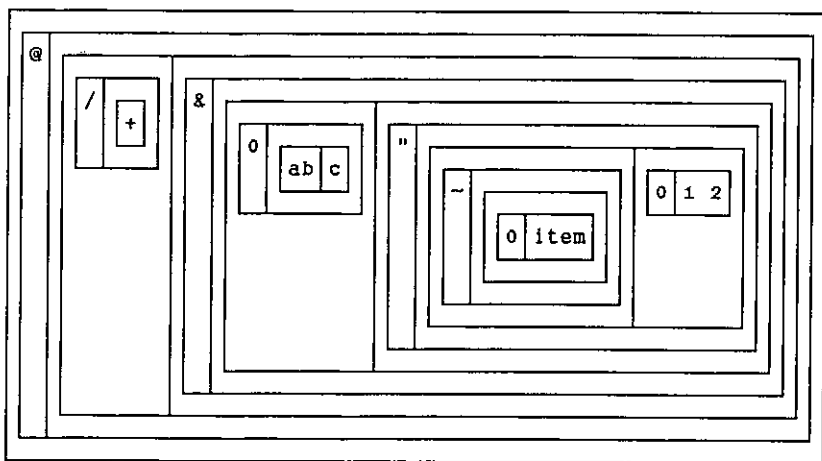
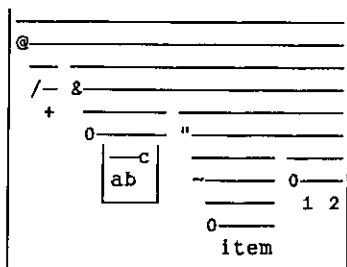


Substitute a span and a cup for the anomalous three-box configuration of an explicit verb.



Spanned display representation may be more ambiguous than unspanned display representation. If display representation is spanned and atomic representation is not, the latter could serve to clarify the former. This would not inconvenience the user since the atomic representations in a failing gerund, for example, are debugged with display representation.

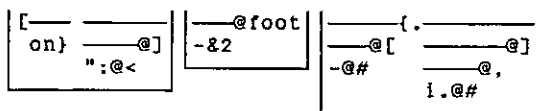
*Typesetter's note:* Word for DOS has long had an annoying habit (probably a carry-over from FX-80 days) of substituting ordinary dashes for the horizontal line-drawing character at print-time. This is not (as far as I can see) part of any of the printer drivers, it is embedded deep in the code! Richard had awful trouble getting this to print nicely ... I am not sure what the best way out is. My fix is to replace all long horizontals with the em-dash character '—' and make sure the appropriate character in my PostScript font is set to draw something identical to a continuous line. Maybe Word 6.0 will fix it, but very likely it won't, in which case I can't offer any sensible suggestions for those of you with Laserjets. Sorry, Adrian.



Spans are neutral on noun lists.



Verbs are the issue. Verbs need optimum display. A tool can easily throw a tight row of spanned display representations on the screen.



# Building a TrueType APL Font for Vector

by *Adrian Smith*

## Motivation

First, let me state a prejudice — I think the world would have been a better place had TrueType never been invented. PostScript already provides an excellent standard for rendering text onto paper, and with the advent of Adobe Type Manager (ATM) under Windows 3.0, it ensured that what you saw on the screen was exactly what you got on the paper. ATM also does a marginally better job at rendering awkward fonts (such as *APL*) on screen at small sizes.

PostScript also has a long tradition of naming fonts, for example Vector is typeset in *Palatino-Roman 13/16* (before reduction). I object violently to having to select something called "*Book Antiqua*" in Winword to get an apparently identical font, but one which is sure to have at least one subtle difference to get around copyright problems! For example, compare the "R" in Helvetica and Arial ... !

Unfortunately, Microsoft are bigger than I am, so (like 7 million other Windows users) I have to go along with them. It is obvious that an increasing proportion of Vector material will be prepared in either Windows Write or Winword; Vector production remains in Word 5.0 under DOS ... so I need a solid and simple way of getting Windows material across into DOS.

## The Options

I could simply move over to Winword. At the moment, a sufficient volume of material is still in DOS format (typically *APL\*PLUS/PC* workspaces) that I would have similar problems moving material across. Also, I find Word 5.0 at least 30% faster for basic mechanical typing and formatting. Vector is quite a big document, and DOS word (in text mode) can scoot around it much faster. Also, Windows has stolen a lot of the most useful keys — for example to make a subheading I simply hit <Alt+s> in Word 5.0; in Winword it would be <Ctrl+Shift+s> which takes that little bit longer.

I could write lots more little translation routines. Jonathan's review of ISI APL was (not unreasonably) created using the ISI TrueType font, and hence the ISI  $\square AV$ . It was about half an hour's work to make a  $\Delta ISIAV$  vector, read the whole review into APL, and spit it back out translated. The problem with this approach is that you rapidly start getting confused and applying the wrong translations ... which makes the APL bits look very funny indeed.

I could try to guide the APL world towards submitting material in the format that is easiest for me. There is only one way to do this — I have to make it easier to use my  $\square AV$  than anyone else's. That means I need to provide both a TrueType APL font encoded how I want it, and some kind of simple APL typewriter so that you can enter code without the hassle of  $\langle \text{Alt}+0251 \rangle$  to get a  $\rho$ .

## The Joys of Font Design

The APL stuff in Vector is printed with an Adobe Type-3 font, which is designed to be human-readable as well as computer readable. For example  $\neq$  is simply defined as:

```
/sf { 3 1 roll moveto 0 rlineto} def % Serifs
/NE { 125 400 400 sf
      125 200 400 sf
      175 80 moveto 475 520 llineto
      stroke } def
```

... the crucial thing is that PostScript allows you to stroke the font with a pen of given width. You also get nice rounded line ends [1] simply by setting the linecaps property:

$\neq$

Converting this to TrueType is not easy! This time, you are only allowed filled outlines, so you have to draw around the character boundary (crossing lines are strictly verboten) and do those nice round ends yourself. It also turns out to be very important to work on a grid such that when the TrueType engine draws your character on the screen, things like  $\bar{\quad}$  don't fall through the cracks in the pixel rounding and vanish from sight.

I did this by asking GoScript to render a complete sample font at 72-point, and output the result to a .PCX file (rather than to paper). I then loaded the resulting file into CorelTRACE and asked it to convert the bitmaps to vector outlines by edge-tracing them. This gave me a reasonable starting point, but obviously it introduced a certain amount of spurious fluff. The tedious bit was to take these outlines, snap them to a standard grid, remove the fluff, and output each character to TrueType. CorelDRAW helped a lot with this process, for example you can draw your own shapes on the 'guidelines' layer and get other characters to snap to them. I worked throughout with an underlying  $\square$  to ensure correct geometry and centering, and an underlying  $I$  to check the obliqueness of the alphabets.

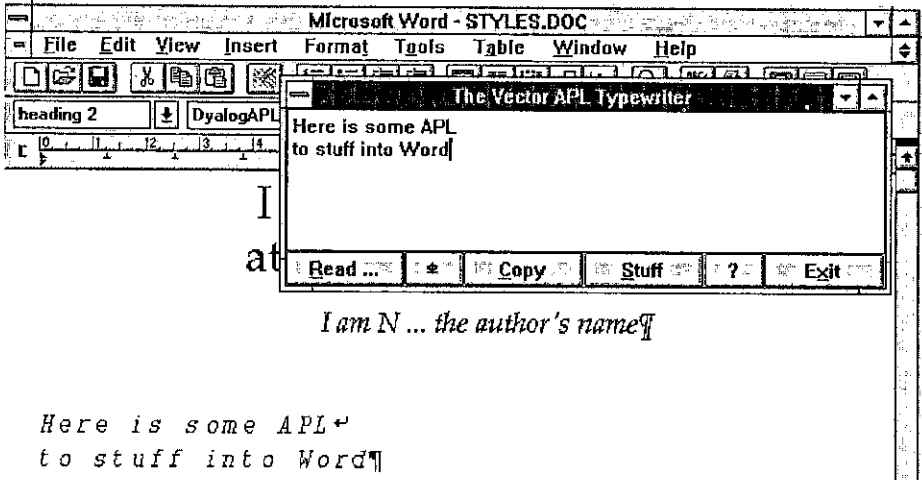
Once you have a TrueType font, you are on the downhill slope. The next package I can recommend is FontMonger (Ares Software: USA+415 578 9090, cost \$145; available in the UK from Camelot mail order: 0800-565656, cost £79+VAT) which is not to be confused with AllType (which has four legs and barks and is not recommended at all). FontMonger will read and write fonts in almost any known format, and it does a pretty good job of adding 'hints' to TrueType to improve the rendering on screen. It makes remapping a font a breeze — you simply start a new font and cut-and-paste from an existing layout. Here is the Vector font as shown on Fontmonger's edit screen:

VECTORPLRF																
Font: VectorAPL Normal																
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
6	·	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
	096	097	098	099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	vec
	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
B	←	→	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔	↔
	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
9	←	△	×	△	□	▽	□	⊞	ε	⋆	φ	ε	ψ	τ	⋆	
	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
A	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞
	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
B	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞
	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
C	L	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞
	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
D	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞	⊞
	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
E	α	β	γ	δ	ε	ζ	η	θ	ι	κ	λ	μ	ν	ξ	ο	π
	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
F	≡	Δ	Σ	≤	≠	±	÷	ω	ο	ν	ρ	υ	⊞	⊞	⊞	⊞
	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255

This is almost the APL\*PLUS ⊞AV — except that anything below chr(32) has been kicked upstairs above chr(127). It works well on paper (even I can't easily spot the difference between it and the APL2741 original) and is quite usable on the screen.

## A Prototype APL Typewriter

The requirement seems to me to be quite simple: a little edit box which will sit on top of Word and let you type APL snippets with a sensible unified keyboard mapping. The box should be capable of importing APL code from ASCII files (translating from all known  $\square AV$  orderings) and should allow you to copy the contents to the Windows clipboard, so that you just click back to the underlying word processor and hit paste. In fact for Winword it can go one better — you can use DDE to stuff text directly into a document at the current cursor position. Something a bit like this, in fact:



Note that this prototype has an  $\pm$  button — so you can type  $\pm 12$ , press <Execute> and get 1 2 3 4 5 6 7 8 9 10 11 12 on the following line automatically. I need to check this with Dyadic, as it may be a bit close to the edge of what you are allowed to ship with a runtime interpreter! It also requires Dyalog 6.3, as an essential part of the design is to have the APL typebox visible at all times, even when Winword (or Write) is running full-screen. This limits availability to "RSN", in other words, some short while after Dyalog release a pukka copy of version 6.3.

*Subject to the above, to get your copy of the font and APL typebox, please send a blank formatted disk (at least 1.2Mb) to Vector Production. Don't forget to include your name and address!*

## Reference

- [1] Joey K. Tuttle, "APL pi — Designing an APL Type font", APL81 Conference Proceedings, pp298 — 302



## RENAISSANCE DATA SYSTEMS

Enlightenment Thru Information Processing

Specializing in Books and Software on APL, J  
and other Curiosities of merit.

### HAVE YOU GIVEN A COPY OF IAPL OR J TO A TEACHER OR STUDENT?

Name: \_\_\_\_\_ Date: \_\_\_\_\_

Street: \_\_\_\_\_

City: \_\_\_\_\_ State: \_\_\_\_\_ Postal Code \_\_\_\_\_

Country: \_\_\_\_\_ Telephone \_\_\_\_\_

For a copy of our most recent catalog,  
please send a self-addressed, legal-  
sized envelope (stamped if from U.S.)  
or mail this form to:

Renaissance Data Systems  
Park West Finance Station  
P. O. Box 20023  
New York, New York 10025-1510

*Learn APL*

*APL Shareware/Demos and Related Publications*

*APL and Mathematics*

*APL AS A TOOL OF THOUGHT Proceedings*

*Other Important Sources of APL Information.*

*APL History*

*APL References and Techniques*

*Special Subjects in APL*

*APL Interpreters and Software*

*J=: A Powerful Dialect of APL*

*Other Software for Use with APL*

*To Russia with Love and for APL92*

*Other Curiosities of Merit*

*I-APL: An international voluntary project of individuals and  
companies seeking to share their love of APL.*



## Index to Advertisers

Dyadic Systems Ltd	2
Manugistics	54
MicroAPL	6
APL Booklist (Renaissance Data Systems)	142

All queries regarding advertising in VECTOR should be made to Gill Smith, at 04393-385.

---

### Submitting Material to Vector

The Vector working group meets towards the end of the month in which Vector appears; we review material for issue n+1 and discuss themes for issues n+2 onwards. Please send the text of submitted articles (with diskette as appropriate) to the Editor:

Jonathan Barman,  
Hill Top House,  
East Garston,  
NEWBURY, Berks RG16 7HD  
Tel: 048839-575

(not after 10.00pm please!)

Authors wishing to use Windows Write should contact Vector Production for a copy of the Vector APL TrueType font. This is available encoded to the Dyalog APL, APL2/PC or (modified) STSC standards: STSC is currently the preferred layout, but either of the alternative layouts are acceptable.

Camera-ready artwork (e.g. advertisements) and diskettes of 'standard' material (e.g. sustaining members news) should be sent to Vector Production, c/o Adrian Smith, Brook House, Gilling East, YORK Tel: 04393-385 (6.00pm - midnight).

## BAA: Membership Application Form

Membership of the British APL Association is open to anyone interested in APL. The membership year runs from 1st May to 30th April.

Name: \_\_\_\_\_  
 Address Line 1: \_\_\_\_\_  
 Address Line 2: \_\_\_\_\_  
 Address Line 3: \_\_\_\_\_  
 Post or zip code: \_\_\_\_\_  
 Country: \_\_\_\_\_  
 Telephone Number: \_\_\_\_\_

Membership category (please tick box): . . . . . 92/93

(these rates also apply to renewals)

UK private membership . . . . .	£12	<input type="checkbox"/>
Overseas private membership . . . . .	£20	<input type="checkbox"/>
Airmail supplement (not needed for Europe) . . . . .	£8	<input type="checkbox"/>
Corporate membership . . . . .	£100	<input type="checkbox"/>
Corporate membership overseas . . . . .	£155	<input type="checkbox"/>
Sustaining membership . . . . .	£430	<input type="checkbox"/>
Non-voting UK member (student/OAP/unemployed only) £6		<input type="checkbox"/>

### PAYMENT – in Sterling 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 or Visa number.

I authorise you to debit my Visa/Mastercard account

Number: \_\_\_\_\_ Expiry date: \_\_\_\_|\_\_\_\_

for the membership category indicated above,

- annually, at the prevailing rate, until further notice  
 one year's subscription only

*Data Protection Act:  
 The information supplied may be  
 stored on computer and processed  
 in accordance with the registration  
 of the British Computer Society.*

(please tick the required option above)

Signature: \_\_\_\_\_ Send the completed form to:

British APL Association, c/o Rowena Small, 8 Cardigan Road, LONDON, E3 5HU

## 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.

### 1992/93 Committee

Chairman:	David Eastwood 071-922 8866	MicroAPL Ltd., South Bank Technopark, 90 London Road, LONDON SE1 6LN
Secretary:	Anthony Camacho 0272-730036	11 Auburn Road, Redland, BRISTOL, BS6 6LS
Treasurer:	Nicholas Small 081-980 7870	8 Cardigan Road, LONDON E3 5HU
Journal Editor:	Jonathan Barman 0488-648575	Hill Top House, East Garston, NEWBURY, Berks RG16 7HD
Activities:	Vacant Post	
Education:	Dr Alan Mayer 0792-295296	European Business Management School, Swansea University, Singleton Park, SWANSEA SA2 8PP
Technical:	Duncan Pearson 0904-603510	c/o Operational Research, Nestlé-Rowntree, YORK YO1 1XY
Projects:	George MacLeod 0442-878065	Greymantle Associates Ltd., Bartrum House, Ravens Lane, BERKHAMSTED, Herts HP4 2DY
Publicity:	Misha Jovanovic 0753-853141	99 Oxford Road, WINDSOR, Berks SL4 5DX
Recruitment:	Dr Peter Branson 081-848 8989	Electronic Data Systems, Stockley Park, UXBRIDGE, Middx UB11 1BQ
Administration:	Rowena Small 081-980 7870	8 Cardigan Road, LONDON E3 5HU

### Journal Working Group

Editor:	Jonathan Barman	0488-648575
Secretary:	Anthony Camacho	0272-730036
Production:	Adrian & Gill Smith	Brook House, Gilling East, YORK (04393-385)
Advertising:	Gill Smith	Brook House, Gilling East, YORK (04393-385)
Support Team:	Ray Cannon (0252-874697), Richard Weber, Sylvia Camacho, Duncan Pearson, John Scarle, David Ziemann (071-267 8032)	

Typeset by APL-385 with MS Word 5.0 and GoScript

Printed in England by Short-Run Press Ltd, Exeter

## VECTOR

VECTOR is the quarterly Journal of the British APL Association and is distributed to Association members in the UK and overseas. The British APL Association is a Specialist Group of the British Computer Society. APL stands for "A Programming Language" — an interactive computer language noted for its elegance, conciseness and fast development speed. It is supported on most mainframes, workstations and personal computers.

## SUSTAINING MEMBERS

The Committee of the British APL Association wish to acknowledge the generous financial support of the following Association Sustaining Members. In many cases these organisations also provide manpower and administrative assistance to the Association at their own cost.

Compass R&D Ltd  
15 Frederick Sanger Road  
Surrey Research Park  
GUILDFORD, Surrey GU2 5YD  
Tel:0483-302249  
Fax:0483-302279

HMW Trading Systems Ltd  
Hamilton House,  
1 Temple Avenue,  
LONDON EC4Y 0HA  
Tel:071-353 4212  
Fax:071-353 3325

MicroAPL Ltd  
South Bank Technopark  
90 London Road  
LONDON SE1 6LN  
Tel:071-922 8866  
Fax:071-928 1006

APL People  
The Old Mailhouse  
Clarence St. BATH, BA1 5NS  
Tel:0225-462602  
Fax:0225-444552

Dutch APL Association  
Postbus 1341  
3430BH Nieuwegein  
Netherlands  
Tel:03474-2337

Kestrel Consulting  
Business & Technology Centre  
Bessener Drive  
STEVENAGE, Herts, SG1 2DX  
Tel:0438-310155  
Fax:0438-310131  
Email: kestrel@apl.demon.co.uk

Dyadic Systems Ltd  
Fiverside View, Basing Road,  
Old Basing, BASINGSTOKE,  
Hants. RG24 0AL  
Tel:0256-811125  
Fax:0256-811130

Cocking & Drury Ltd  
180 Tottenham Court Road  
LONDON, W1P 9LE  
Tel:071-436 9481  
Fax:071-436 0524

Reuters Ltd  
Reuters Nederland BV  
PO Box 8230, 1005 AE Amsterdam  
Netherlands  
Tel:+31 20 570 8733  
Fax:+31 20 570 8758

Peter Cyrix Systems  
22 Hereford Road  
LONDON W2 4AA  
Tel:071-229 5344

Mannigistics  
2115 East Jefferson St  
Rockville  
MARYLAND 20852 USA  
Tel: (301) 984-5412  
Fax: (301) 984-5094