VECTOR

FEATURING EXPERT SYSTEMS

British APL Association

A Specialist Group of the British Computer Society

Vol.1 No.3 January 1985

CONTRIBUTIONS

All contributions to VECTOR should be sent to the Editor at the address given on the inside back cover. Letters and articles are welcomed 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 submitted in duplicate and accompanied by as much visual material as possible, including a photograph of the author. Unless otherwise specified each item will be considered for publication as a personal statement by its author, who accepts legal responsibility that its publication is not restricted by copyright. The provision of camera-ready or machine-readable copy is encouraged: please contact the Editor beforehand. Program listings should indicate the computer system on which they have been run. APL symbols should be displayed on a separate line and not embedded in narrative. Except where indicated, items published in VECTOR may be freely reprinted with appropriate acknowledgement.

MEMBERSHIP

Category	*Fee p.a.*	*VECTOR copies*	*Passes*
Nonvoting student membership	Free (1984)	1	1
UK Private membership	£ 6	1	1
Overseas private membership	£ 10	1	1
Corporate membership	£ 50	10	5
Sustaining membership	£250	100	5

The membership year runs from 1st May to 30th April. Applications for membership should be made on the form at the end of the journal. Passes are required for entry to some Association events and for voting at Annual General Meetings. Applications for student membership will be accepted without charge at the discretion of the Committee on the production of educational bona fides and a recommendation from a course supervisor. Overseas membership rates include VECTOR airmail postage and should be paid in UK £.

Corporate membership is offered to organisations where APL is in professional use. Corporate members receive multiple copies of VECTOR and are offered group attendance of Association meetings. Partaking individuals need not be identified but a contact person should be nominated 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 the editorial section of the journal and opportunities to inform APL users of their products via seminars and articles.

ADVERTISING

Advertisements in VECTOR should be submitted in typeset camera-ready A5 portrait format with a 20 mm blank border. Illustrations should be black-and-white photographs or line drawings. Rates are £200 per page. A6 and A7 sizes are offered pro-rata subject to layout constraints. Deadlines are:

Advertisement booking:	3rd Friday in April, July, October & January.
Camera-ready copy:	1 week later. Distribution: 2 weeks later.

CONTENTS

VECTOR 3 was edited and assembled by Adrian Smith. Adrian is still at Rowntrees, in spite of the unfortunate incident in the Polo Department.

Thanks are due to Gill Smith, Wendy Hoare and Mark Jones for their invaluable help in checking the proofs.

What's new in 85 — the reception committee prepares for the big day.

and recovers afterwards!

GUEST EDITORIAL: THE EVOLUTION OF APL

by Adrian Smith

Evolution has emerged largely by accident as a strong theme in three successive issues of VECTOR. It is a topic which often defies intuition; a domain where seemingly circular arguments turn out oddly and disturbingly valid. What follows is a search for an evolutionary perspective of APL.

Until about 2 years ago APL was quite clearly a single distinct species. True there were a few interesting mutations, but nothing which in any way prevented the free exchange of ideas. It is however a sad truth that evolution *within* a species is a desperately slow business. Even given the marvellous genetic diversity of APL (how many ways do *you* know of chopping the trailing blanks off a string?) the worldwide pool of accepted idioms has almost ceased to grow.

Many interesting mutations have come and gone, often to re-appear and vanish once more. The idea of adding vectors row or columnwise to matrices is one such, and illustrates a pattern which is absolutely typical of natural systems. In a stable environment mutations are almost inevitably penalized; white rabbits get eaten!

How then does the natural world progress? Sometimes through white rabbits, but more often through isolation, speciation and (I regret to say) extinction. First of all you must take a select few of your creatures and isolate them well away from predators and competing groups. They rapidly become inbred, and all sorts of weird mutations may survive. Given several such 'peripheral isolates' it is inevitable that a 'super rabbit' will eventually emerge; it may even be white!!

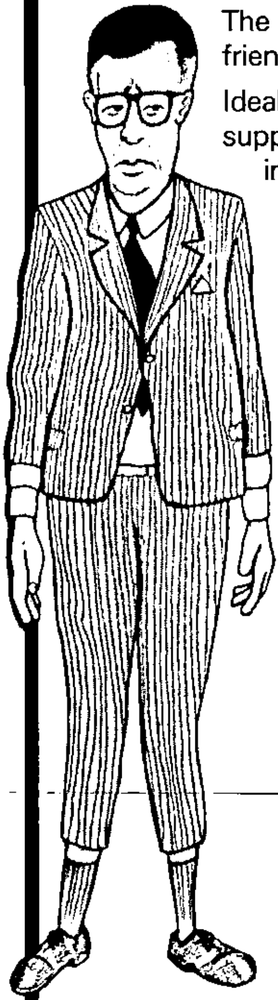
The conditions are ripe for a take-over: re-introduce your Mark-II Coney and it will wipe out the indigenous Mark-I in less time than it takes to say 'Charles Darwin'. Indeed the invasion often comes as a multi-pronged attack; several island communities may generate new species, each better adapted to some specific ecological niche than their single precursor, none competing directly with each other. Enough said.

Let's look at VS APL (the precursor), and NARS, IPSA, APL2 and the rest (the peripheral isolates) in this light. The inbreeding was quite intentional, and the mutations have been many and varied. Now we have the invasion, and it comes from many sides. As these 'nested-array' species fight it out over the high ground, a shoal of keyworders appears to contest the micro scene. Will VS APL be throttled to extinction by these improbable allies? Here is my guess; it is as likely to be wrong as anyone else's.

I think that APL2 will kill off all the other nesters very soon. NARS and IPSA will retreat to their island fastnesses, there to perish or flourish in ever greater isolation. It matters little. I think that keyword APL has about as much in common with VS APL as a Chihuahua has with a Wolfhound, but it is not truly a separate species. We *can* read and appreciate each others' code, and the products of that union may at last let APL take flight. Whether the final result uses words or funny symbols (or more likely both in continued co-existence) is pretty irrelevant. The ideas and power of APL go a lot deeper than a fancy typeface, and I would happily take the keyword route simply to avoid messing up my keyboard with stickers!

IBM will do their utmost to wipe out VS APL (and the idea of keywords) with APL2. They will struggle long and hard, and I think that ultimately they will fail. APL2 cannot beat IPSA and NARS on merit; it can and will beat them on weight. That's how Brontosaurs evolved; I think I'll back Archaeopteryx.

MetaSys... WHEN A STANDARD PACKAGE WON'T FIT



The new custom software service for reliable, user-friendly systems built to individual requirements.

Ideal for planning, modelling, forecasting & decision support — or wherever packages are unavailable or inadequate.

Combines the powerful APL language (to cut development time & cost) with compiled C or Assembler (for fast run-time).

You see a prototype of your system (usually within a few days) to ensure that the system we develop is the system you want.

Includes full HELP facilities, menus, full screen, flexible reports, in-program calculator, mainframe communication, etc.

Full back-up and support service.

To discuss your requirements without commitment, just phone us on 01-670 7959.

MetaSys... A professional custom solution at a competitive price.

For IBM PC XT AT, COMPAQ & compatibles, UNIX & XENIX

MTS **MetaTechnics Systems Ltd**
Unit 216, 62 Tritton Road, London SE21 8DE, 01-670 7959

The Systems Builders

GENERAL CORRESPONDENCE

Following the pattern set in Issue 2 of VECTOR we are continuing to split letters into general and technical sections. It would be most helpful if correspondents would indicate which section their letter is intended for. Letters containing APL symbols will normally be considered Technical; please note the requirements on the inclusion of APL code stated on the inside cover. The editor reserves the right to edit letters unless a writer states that a letter is to be published in full or not at all.

WHOOPS

From Mr. J. Sullivan 7th November 1984

Sir: In his article 'XPL: An Expert Systems Framework in APL' in vol 1 No. 2 Robert Bittlestone assigns the authorship of 'Do Androids Dream of Electric Sheep?' to Ray Bradbury. It was, of course, written by Philip K. Dick in 1968 and later formed the basis for the motion picture 'Blade Runner'. The British publishers were Rapp & Whiting (1969) and Granada (Panther 1972): I do not know who the American publishers were.

Yours sincerely

J. Sullivan,
Business Development Section,
National Westminster Bank plc.,
41 Lothbury,
London.

DEC Mnemonics

From John Garland 1st November 1984

Sir: In both issues of VECTOR there has been comment and correspondence about the symbols used to represent APL operators. Probably through ignorance as a result of being neither a computer scientist nor a computer programmer I have wondered what all the fuss is about.

I use an 'unofficial' version of APL-11 on a DEC VAX 11/780 via a VT100 terminal. APL-11 uses a mnemonic representation of the APL operators; for example drop is .DA (down arrow), del is .DL, iota is .IO and so on. These mnemonics are quickly learned and I must admit that typing .GU for grade up seems to me to be easier than typing the corresponding APL version which requires a fussy backspace.

It thus seems that the original and mystifying symbolism of APL is largely unnecessary for a successful use of the language, and its replacement with commonsense mnemonics such as those used by DEC (see APL-11 Programmers Reference Manual; Jan 1980: pages 1-5 et seq) might do much to liberalise the language and hasten its more widespread adoption. It would also presumably do away with the need for special keyboards.

Yours sincerely,

J.H.N. Garland,
87 Whitney Drive,
Stevenage,
Herts.

Nowhere Fast

From Bob Pullman 31st August 1984

Sir: VECTOR is a great idea. I hope it gets the subscription support it deserves. The rank and file of APL have been left out in the cold by the APL Mafia (would VECTOR really 'name the guilty men'?) Not to take anything away from the people who have poured a lot of time, gratis, into APL, but:

1. APL/n has mostly the same speakers as APL/n – 1
2. Almost all of the action is behind closed doors.
3. The info passed down to local SIGAPLs is irrelevantly late or non-existent.

VECTOR proves that there are authors other than the names that occur in proceedings, and that there are newsworthy events in the APL world. My fledgling company has joined the APL Young Turks, the 3rd world of APL. We basically believe that Sharp, STSC, IBM etc have bungled things and are taking APL nowhere fast. There are a lot of small APL firms, but no one except Phil Van Cleave has tried to mount a challenge to the old guard. I don't think there's any doubt that IBM, STSC & Sharp manipulate the conferences and publications. Thus I see VECTOR as the first publication to present the other voices of the APL world.

One way of telling for sure is to survey your readers to assess the feelings of the man-in-the-street. From reading V1 I know you know what the questions are and aren't afraid to ask them. Everyone that I know feels the way I do.

Yours sincerely,

Bob Pullman,
164 Pinnacle Road,
Rochester,
NY 14620.

QUICK-REFERENCE APL DIARY

We do our best to ensure the accuracy of these listings before publication, but with the change of venue (see below and also the news page) we would strongly advise readers to check the details nearer to the proposed date. Members are sent advance notice of all meetings.

Unless otherwise indicated, meetings are free, open to non-members, and take place at 2.00 p.m. at:

Royal Overseas League
Park Place
(off St. James's Street),
London SW1.

This is two minutes walk from either Green Park or Piccadilly Underground stations, and offers both a restaurant and a bar. There will normally be a bookstall at all the London meetings; attendees can buy from a selection of hard-to-find APL literature.

Date	**Venue**	**Event**
March 15	London	Graphics: IBM and GKS
April 26	London	Nested Arrays Workshop
May 12-16	Seattle	APL'85: APL and the Future.
May 28 (Tuesday)	London	AGM Mainframes, Micros and Co-existence.
September 13	London	Consultants Question-time
October 18	London	Large APL Applications
November 15	London	Compilers and Interpreters
1986 July 7-11	UMIST	APL-86 Topics include: Teaching APL; APL design; Relational Database Demystifying APL

Further details are given in the Association News section in the following pages. A detailed prospectus for APL-86 will appear in the next issue of VECTOR.

Details of overseas events are in the International News section.

BRITISH APL ASSOCIATION NEWS

Situations Filled

We are delighted to welcome Geoff Kemish as our new Technical Officer. Geoff is unfortunately away in the USA for the whole of January, but we shall make every effort to obtain his picture in time for the next issue!

Change of Venue

Royal Overseas League
Park Place
(off St. James's Street)
London SW1

After many years holding meetings at Imperial College we have decided that a change of venue is both timely and appropriate.

The Royal Overseas League is centrally situated, two or three minutes walk from Piccadilly and Green Park Underground Station; it provides comfortable meeting rooms and has both a restaurant and a bar. There is also a wide selection of eating and drinking establishments nearby.

In addition to the formal aspect of the meetings there is ample opportunity to either discuss technical matters or to just socialise with other people from the APL community; part of our intention in making this move has been to extend the scope for these casual discussions. One of the main objectives which we have as an Association is to extend the APL community spirit, so if you haven't been to one of the meetings for a while come and re-establish contact in the new surroundings.

Meeting Dates

A consequence of moving is that we have had to shift our meeting dates a little; this will settle down again over the next few months and we will be making information available within VECTOR, by direct mailshots and the Computing Diary Page.

March 15 - Graphics and APL

This meeting will cover different ground and approaches to graphics from the January 1984 meeting with which it shares a title; present plans are to emphasise the GKS graphics standard and the IBM path to graphics.

ACTIVITIES PROGRAMME 1985

The following technical meetings are planned during 1985:

March 15	Graphics; IBM graphics and the GKS graphics standard Again expanding on the theme of a previous meeting, this time we look at APL graphics from an IBM viewpoint and at the increasingly-popular GKS graphics standard (a chance for APL to align itself with the rest of the computing world?)

April 26	Nested Arrays Workshop APL2 is well on the way; some other implementations have had some of its features for several years. We explore the impact which generalised arrays have on the application developer, including case studies.
May 28	AGM plus Mainframes, Micros and Co-existence The ever-popular opportunity to show your appreciation of your Committee; followed by the relatively soothing topic of how to have APL working cooperatively on both mainframes and micros.
September 13	Consultants Question-time Are they really worth those fees? Come along and find out.
October 18	Large APL Applications and Systems APL applications are broadening from the one-man show, what are the implications? Does it mean that APL has to adopt the working disciplines of the data processing department? And what are these disciplines?
November 15	Compilers and Interpreters Since its inception APL has been interpreted; many have yearned for compiled APL. Now that the option is available, what impact does it have? Plus a look under the interpreter bonnet and a study of how compiler and interpreter internals differ.

The programme is subject to revision; further details of each meeting will be sent to BAA members nearer to the date of the meeting. Unless otherwise stated all meetings take place in London during the afternoon.

We would hope that there will be at least one BAA Special Event during the year; an announcement will be made in due course.

1986	APL86 of course, our regular technical meetings will continue, among the prospective topics are APL Design, Teaching APL, APL and Relational Databases and Demystifying APL.

Those wishing to offer either possible subjects or themselves as potential speakers are urged to contact the Activities Officer.

NOTICE ABOUT ELECTIONS TO THE COMMITTEE

Elections have always been held by simple majority vote of those attending the AGM and Officers have always been elected for a year. Your committee has discussed changes to this arrangement and decided not to recommend any change for the 1985 elections. (The reasons for change would be to be fairer to the distant members by allowing them to vote without having to attend, and to protect the Association against a complete change of Officers which would lose continuity).

Approximately two thirds of our members have attended a meeting in London during the last year, so the committee did not feel justified in changing the election arrangements for 1985 but instead has decided to consult members at the AGM to see whether changes are desired (those not attending may comment by writing to the Secretary).

As an example of the second reason, to plan and run APL86 under one management your committee has had to appoint an APL86 subcommittee which although subsidiary to the Association committee, will hold office until APL86 is over and the accounts cleared up (i.e. certainly until the autumn of 1986). There will be a proposal at the AGM that half the Officers are elected for two years, and that each year from 1986 onwards half the Officers are elected.

The committee intends that members should be well informed about candidates at the next election. Details of candidates are to be circulated three weeks before the election. Every candidate must be proposed and seconded and must provide a brief description (about 100 words) of their suitability and their intentions should they be elected. Every candidate is required to have paid an individual membership fee (currently £6)and to recognise that it is a duty of a committee member to attend every committee meeting and to perform the appropriate tasks.

The posts to be filled by election at the next AGM are: Chairman (who is required to be a member of the British Computer Society), Treasurer, Secretary, Activities Officer (organiser of meetings and special events), Journals Officer (chief editor and organiser of VECTOR), Publicity Officer (who is to arrange for promotion and coverage of the Association's activities), Technical Officer (who organises technical vetting and reviews of products, articles and papers) and Education Officer (who is to promote education in APL and the use of APL in education). Each officer is encouraged to form a working party of members to help with the job and to provide a deputy to report at any committee meeting that circumstances make it impossible to attend in person. All officers are asked to help with the Association's work in other ways.

None of the foregoing precludes last minute proposal and seconding (at the AGM) of an individual member in good standing for any office. Please send details of all candidates to be circulated in advance, to the Secretary, 2, Blenheim Road, St. Albans, Herts AL1 4NR to arrive by the end of April 1985.

APL86

As you will see from the following Call for Papers, the APL86 Conference and Exhibition will be held at the University of Manchester Institute of Science and Technology (UMIST) from 7-11 July 1986.

As most of you will know, we had planned to hold APL86 at Cambridge. Unfortunately the growing requirements for exhibition space and the conference outgrew the available facilities, and we have had to have a new venue.

UMIST, however, has excellent facilities for a major international conference such as APL86 with a number of modern lecture theatres sited around the exhibition floors. The University, which has a long tradition in computing is in the heart of Manchester, England's second city, with excellent road and rail access and a major international airport.

More details about APL86 will appear in the next issue of VECTOR, and there will be an APL86 stand at the forthcoming APL85 conference in Seattle. If you might be interested in exhibiting hardware or software at APL86 please ensure that you are on our exhibitors' mailing list by writing to:

APL Exhibition Organiser,
BISL Conference Department,
British Computer Society,
13 Mansfield Street,
London W1M 0BP.

CALL FOR PAPERS

International Conference on APL
and its applications

University of Manchester Institute of Science & Technology
Manchester, England.

July 7th to 11th, 1986.

IMPORTANT DATES:

Preliminary Submission	June 1, 1985
Submission Deadline	November 1, 1985
Acceptance Notification	January 10, 1986
Final Version	April 4, 1986

APL86 will be the first of the annual series of international conferences on the development and applications of APL to be held in England. It will take place at the University of Manchester Institute of Science and Technology from July 7th — 11th, 1986. There will be a simultaneous exhibition by suppliers of APL-based services and products.

The conference will emphasise practical aspects of APL, and "live" presentations to accompany submitted papers will be encouraged by the provision of a wide range of computing and audio-visual equipment.

As is traditional at the APL Conferences, a number of distinguished speakers from both within the APL community and other areas, such as Expert Systems, will be invited to present papers, and lead longer discussion sessions, based around "live" presentations.

The conference will also include a number of tutorial sessions, of interest to both newcomers to APL, and those with more experience in the use of the language.

The conference will have three main themes:

- APL in action, including sessions on the use of APL in Personal Computing, Information Centres and Education. Other sessions currently planned include, for example, Full-screen and Graphics orientated systems, applications of APL in Expert Systems, and programmer productivity tools.
- APL and its environment, including sessions covering, for example, the interactions between APL and its host Operating Systems, APL as a compiled language, APL-dedicated hardware configurations, and the use of APL as an "engine" in systems constructed largely in other languages.
- The APL notation, including sessions covering, for example, experience with the ISO Standard, and with "second-generation" APLs, the influence of array theory on further extensions, the integration of the "second-generation" extensions, and APL as a teaching notation.

The conference Committee are keen to encourage those who might have not previously considered submitting a paper to such a conference to share their experiences with others in the APL community.

Submission of Papers

Those intending to submit papers on one of the conference themes must make at least a preliminary submission by June 1st 1985; including a title, an abstract, and an indication of the nature of the accompanying "live" presentation and of the equipment that would be required.

Full draft papers, for consideration by referees, must be submitted by November 1st 1985. Three copies of the paper, which is to be in English and should not exceed 6000 words must be submitted, together with an abstract and a list of keywords for subject identification and classification, and a fuller description of the "live" presentation. One or more referees may request that the presentation is made as part of the judging process.

Notification of the referees' decisions will be sent out on January 10th 1986. Authors of papers accepted must provide final copies of their papers suitable for photo-reproduction by April 4th 1986.

All papers accepted and received by the final submission deadline will be published in the Conference Proceedings. Other papers will be considered for publication in VECTOR, in the four issues following the conference.

It is intended that prizes will be awarded for the "best" papers and presentations given to the conference.

Further details can be obtained from:

BISL Conference Department (APL86)
The British Computer Society
13 Mansfield Street
London W1M 0PB, England

APL People Ltd.

3, Cross Lane Close,
Orwell,
Nr. Royston,
Herts. SG8 5QW

APPOINTMENTS

A placement service solely for the APL community.
Use our extensive knowledge, contacts and experience in the APL world.
We have vacancies NOW, both permanent and contract .... and our confidential advice on career moves is always freely available.

CONSULTANCY

A consultancy service designed to solve your APL problems.
Consultant's Toolkit and Supervision available.

Current projects include:-
* graphics
* project management
* databases
* system design
* system 'rescue'
* documentation
* VSAPL under VSPC, CMS, TSO
  APL*Plus/PC, Sharp APL

To discuss your requirements: — Call Valerie Lusmore on 04626-77375

Registered employment agency licence SE6440

INTERNATIONAL APL NEWS

Compiled by Adrian Smith

This column is open to other APL associations outside the UK; please write to us with your news and we shall be pleased to broadcast extracts for you. Many VECTOR subscriptions are from overseas, and we hope in this way to keep APL users in touch with each other internationally.

We were particularly pleased to receive a copy of the BACUS (Belgian APL Association) News which abstracted and reprinted several articles from Vectors 1 and 2. Many thanks for helping to spread the word!

I.P. Sharp 1984 APL Users Meeting

This was a three day conference based around 'Information Centres and Changing Technology'. It was attended by around 300 delegates (by no means all of them Sharp users) and had a very cosmopolitan flavour. Personally I found it a very worthwhile trip, and my notes are included in the 'Recent Meetings' section of VECTOR.

APL'85 — APL and the Future

It is not too early to be making arrangements for APL'85 in Seattle, Washington — the Emerald City. The following extract comes from the Conference Announcement:

"APL'85, the 1985 Conference of the international APL community will be held in Seattle, Washington, USA — May 12-16, 1985. The Conference is co-sponsored by ACM SIGAPL and by the Puget Sound Chapter of ACM. The Conference theme is: APL And The Future.

"The Conference Committee welcomes you to the scenic Pacific Northwest, home of beautiful mountains, forests, lakes and Puget Sound. May is customarily a warm and sunny time in Seattle.

"The conference site is the Seattle Westin Hotel. Its twin towers contain complete conference facilities, over 800 sleeping rooms, swimming, health and dining facilities. Located in downtown Seattle, the Westin is in easy walking distance of many of Seattle's fine tourist attractions. You can also ride the Monorail to the Seattle Center, then eat at the top of the 600-foot Space Needle, and visit the Center's fountains, shops and cultural attractions. Seattle's FREE downtown bus service will take you to other attractions, such as the waterfront and restored Pioneer Square. For more extensive touring, you can take a Gray Line boat and bus tour of Seattle, or a one- or two-day side trip to Vancouver or Victoria BC.

"The Conference starts Sunday afternoon, and ends Thursday noon. This is a day longer than previous North American APL Conferences, allowing fewer papers at one time so you can hear more papers of your choice. There will be the traditional fine banquet. The extra evening will feature a boat ride on beautiful Puget Sound, with an Indian-style salmon dinner, and entertainment by a Pacific Northwest specialty — African music from Zimbabwe played by a Marimba band.

"An excellent technical exhibit will feature the latest in APL Hardware, Software, Applications and Literature, with an emphasis on Personal Computers."

Invited Speakers

We're not giving away all of our secrets.

Panel Discussions (What would you like to hear?)

Corporate use of APL: Case Studies
Review of APL Micro Systems
Commercial Applications in APL
Information Center Products

Tutorials (What would you like to learn?)

Using APL and GDDM
APL2
Nested Arrays

Special Topics (What would you like to listen to?)

When not to use APL
Writing APL Interpreters
Problem Solving
Economic Justification of APL

Demonstrations (What would you like to see?)

Graphics
Business Applications
Micro APL

To obtain a registration packet, please send your name and address to:

Mike Metzger, 1800 N 46th Street, Seattle, WA 98103

or use IPSA and STSC Mailbox Code — APL85, attention Registrar.

The call for papers closed on September 14th., but other questions about the technical program should be submitted to the Program Chair:

Robert C. Metzger, I.P. Sharp Associates Inc.,
1200 First Federal Plaza, Rochester, NY 14614,
USA (IPSA & STSC Mailbox Code — RCM).

Questions about any other Conference matters should be directed to the Conference Chair:

Bob Gailer, Boeing Computer Services 9A-90,
Education & Training Divn., PO Box 24346, Seattle,
WA 98124, USA (IPSA & STSC Mailbox Code — APL85).

APL Quote Quad Subscriptions

As an information service to VECTOR readers and as a courtesy to the editors of "APL Quote Quad" for allowing us to reprint some of their news material, we are relaying the details of how to subscribe to their journal.

APL Quote Quad is an informal quarterly publication containing articles, algorithms, announcements, book reviews, correspondence, problems, and other materials of interest to users of the programming language APL. Subscription is a benefit of SIGAPL membership, but is also available to non-members. Address inquiries to:

Fred Aronson
Coordinator of SIGs
Association for Computing Machinery
11 West 42nd Street
New York, New York Telephone: (212)869-7440
USA 10036 Telex: 421686 ACMUI

The price of annual subscriptions to 'APL Quote Quad' is US$30.00.

International APL Organisations

We are publishing with permission an updated version of the list of organisations appearing in APL Quote Quad, Vol 14 No. 1. The VECTOR Editor would be pleased to receive information from any of these groups for publication.

Austria
APL-Club Austria
Dr. Siegfried Streit, Chair
Wiener Stadtwerke
Mariannengasse 4-6
A-1095 Vienna
Austria

Belgium **BACUS**
An affiliate of the Computational and Applied Mathematics Group

Belgian APL-CAM Users Society
Joseph De Kerf, Chair
Rooienberg 72
B-2570 Duffel
Belgium

Telephone: 015/314724
IPSA mailbox: KERF

Canada: Toronto **CIPS APL SIG**
An affiliate of the Canadian Information Processing Society

Toronto Special Interest Group on APL
Richard Levine, APL Co-ordinator
Xerox Canada Inc.
703 Don Mills Road
Don Mills, Ontario M3C 1S2
Canada

Telephone: (416)429-6750, Ext. 2684
(after 4 p.m.)
IPSA mailbox: RLEV; group box: TSIG

Finland **FinnAPL**
Finnish APL Association
Gustav Tollet, Chair
P.O. Box 1005
SF-00101 Helsinki 10
Finland

France
A subsidiary of the Association Francaise de Cybernétique, Économique, et Technique
Group APL, AFCET
Gilles A. Martin, Chair
CISI
35 Boulevarde Brune
F-75014 Paris
France

Germany **APLCG**
APL-Club of Germany e.V.
Prof. Dr. Wolffried Stucky
Postfach 6380
D-7500 Karlsruhe 1
Federal Republic of Germany

Italy
A subsidiary of the Associazione Italiana di Calcolo Automatico
APL Club Italia
Giorgio Faconti, President
CNUCE
Via S. Maria 36
I-56100 Pisa
Italy

Telephone: (50)45245

Mexico
Sociedad Méxicana de APL, A.C.
Erick Alvarado, President
Teleinformática de México, S.A.
Arenal 40
México 20, Dist. Féderal
Mexico

Telephone: (905)550 80 33

Netherlands APLWG
A subsidiary of the System-Programming Section of the Netherlands Society for Informatics (NGI)
Werkgroep APL
Dhr. Bram Kornaat, Secretary
Rijks Computer Centrum,
Fauststraat 1
NL-7323 BA Apeldoorn
Netherlands

Telephone: (55)778822

Portugal
Associacao Portuguesa da Linguagem APL
José M. Henriques
Rua Pedro de Barcelos 14
Encosta do Restelo
P-1400 Lisboa
Portugal

Sweden SIG-APL
A subsidiary of the Swedish Society for Informatics
APL Special Interest Group
Bo Lundmark
Sperry AB
P.O. Box 1138
S-171 22 Solna
Sweden

Switzerland SAUG
Swiss APL User Group
Herr Th. Formanek, President
Postfach
CH-3000 Bern 29
Switzerland

Telephone: (31)624980

United Kingdom
A Specialist Group of the British Computer Society
British APL Association
Philip Goacher, Chairman
British Computer Society
13 Mansfield Street
London W1M 0BP, England

Telephone: 01 637 0471

USA: Dallas-Fort Worth SWAPL
Southwest APL Users' Group
Don Hatfield, Secy/Treas.
2809 Apple Valley
Garland, Texas
USA 75043

USA: Denver DIGAPL
Denver Interest Group for APL
P.O. Box 3931
Englewood, Colorado
USA 80155

USA: Houston
Houston APL Special Interest Group
Mary Hall, President
Charter Oil Company
P.O. Box 87535
Houston, Texas
USA 77287/7535

Telephone: (713)640-4724

USA: New York City NY/SIGAPL
New York Chapter, SIGAPL
P.O. Box 524
New York City, New York
USA 10025

USA: Rochester
Rochester Chapter, ACM SIGAPL
Robert Pullman, President
P.O. Box 92487
Rochester, New York
USA 14692

Telephone: (716)244-4732

USA: San Francisco APLBUG
APL Bay Area Users' Group
Dick Foss, Treasurer
Crocker National Corporation
West Tower, 29th Floor
One Montgomery Street
San Francisco, California
USA 94104

USA: Seattle
UW APL Users Group / SIGAPL
J.R. Atkins
University of Washington DH-05
Seattle, Washington
USA 98155

USA: Washington
Washington Chapter, ACM SIGAPL
Lawrence Dusold
Dept. of Health, Education and
Welfare HFF-146
200 C Street SW
Washington, District of Columbia
USA 20204

Telephone: (202)245-1413

ANNOUNCING

dyalog APL

VERSION 3·0

The HIGH PERFORMANCE APL for UNIX* systems

Dyalog APL is FAST

The new version of Dyalog APL is at least twice as fast as before. The component file system and the performance of many primitive functions has been substantially enhanced, so that for some applications even greater improvement will be experienced. Our own benchmarks based on real applications indicate that there is now no faster APL available for supermicros.

Dyalog APL is FUNCTIONAL

Features include nested arrays, full-screen editor, full-screen data manager, component file system, error trapping, external variables, large dynamic workspace.

Dyalog APL is POWERFUL

Dyalog APL's interface to the UNIX* Operating System is unrivalled. Not only can you run any UNIX* program from within the workspace, but you can capture its results in an APL variable for subsequent processing. Furthermore, you can write functions in a compiled language such as C or Fortran.

Dyalog APL is RELIABLE and WELL ESTABLISHED

Dyalog APL is now implemented on over 20 different UNIX* systems and has already been installed in more than 50 companies in eight countries.

For further information contact

Sales Department
Dyadic Systems Limited
30 Camp Road Farnborough Hampshire GU14 6EW
Telephone Farnborough (0252) 547222 Telex 858811

*UNIX is a trademark of A T & T Bell Laboratories

BRITISH COMPUTER SOCIETY NEWS

Compiled by Philip Goacher

The BCS has more than 40 Specialist Groups of which the British APL Association is one. Many of the groups run seminars and conferences during the year which may be of interest to you — these are included in the following list of BCS events in 1985.

March 27-29 Brighton	Current Perspectives in Health Care Computing 85 (Medical Information Specialist Group)
April 9 London	Knowledge Based Programming (Advanced Programming Specialist Group)
April 17-18 Huddersfield	Database 85 (Database Specialist Group)
April 17-19 Bournemouth	Small Computers & Business (South of England Branches Conference)
June 4	Is CAE Communication with the computer really effective? (CAD/CAE Specialist Group)
June 5 London	The Computing of Nursing Manpower and Finance (Nursing Specialist Group)
July 16	Distribution of Management & Management of Distribution (Distributed Systems Specialist Group)
September 16-20 Univ. of East Anglia	HCI 85 (Human Computer Interaction Specialist Group)
December 16-19 Univ. of Warwick	Expert Systems 85 (Expert Systems Specialist Group)

For further details of these events please contact:

BISL Conference Department
British Computer Society
13 Mansfield Street
London W1M 0BP

01-637 0471

NEWS FROM SUSTAINING MEMBERS

Compiled by David Preedy

The category of "Sustaining Membership" of the British APL Association is available to any company trading in APL products and services; it is seen as a formal statement that these companies regard APL as more than simply another programming language, and is intended to promote increased interest and activity in APL and in the Association.

As well as receiving public acknowledgement for their sponsorship of the Association, sustaining members receive bulk copies of VECTOR to distribute to their customers, and are offered the opportunity to submit news material for this section of the Journal. They are also able to submit announcements for Association meetings (subject to the approval of the Activities Officer) and are invited to inform APL users of their products via seminars and articles.

The Committee of the British APL Association would like to acknowledge the generous financial support of the following organisations that have become Association Sustaining Members. In many cases these organisations also provide manpower and administrative assistance to the Association at no cost.

APL People Ltd.	3 Cross Lane Close, Orwell, nr Royston, Herts, SG8 5QW. Tel. 04626 77375
APL*PLUS Ltd.	Aston Science Park, Love Lane, Birmingham, B7 4BJ. Tel. 021-359 5096
CACI Inc. International	Oriel House, 26 The Quadrant, Richmond, Surrey TW9 1DL. Tel. 01-940 3606
Cocking & Drury Ltd.	16 Berkeley Street, London, W1X 5AE Tel. 01-493 6172
Dyadic Systems Ltd.	30 Camp Road, Farnborough, Hants. Tel. 0252-547222
Inner Product Ltd.	Eagle House, 73 Clapham Common South Side, London, SW4 9DG. Tel. 01-673 3354
MicroAPL Ltd.	Unit 1F, Nine Elms Industrial Estate, 87 Kirtling Street, London SW8 5BP. Tel. 01-622 0395
I.P. Sharp Associates	10 Dean Farrar Street, London SW1. Tel. 01-222 7033

APL People Ltd.

APL People's consultancy team, led by Tim Perry and David Alis, has continued to consolidate its studies of APL on the IBM PC which they began in 1982. They have now successfully downloaded several extensive systems from an IBM mainframe using tapes, which have proved much faster and more convenient than directly via the IRMA board for the large amounts involved. These systems were converted to APL*PLUS on the PC and are working very well, allowing far more users than before. They have also developed the PC software to allow the swift transfer, via IRMA, of smaller amounts of data or programs, directly between the systems — and are now transferring the elegant colour graphics software, which excited so much interest at the December APL exhibition, from the PC to a version on a standard IBM mainframe.

On the 'people' side, the employment agency has extended their activities from their original base of supplying consultants on a contract basis, to matching people in APL to the many permanent jobs available. This has proved a popular move within the industry and has expanded so much that we have had to move into temporary office space almost a year ahead of a planned move in 1985. (That is why we retain our familiar address but have a new telephone number.) This side of our activities will be expanding still further in 1985, with more people (*not* computers, please note) involved. Currently new job opportunities and CVs are arriving at about the same rate — we wish they all matched up, but we do our best with □ pegs and ○ holes.

APL*PLUS Limited

APL*PLUS Ltd. announce Release 4.0 of their APL development system for the IBM PC and IBM PC/AT. This allows soft character-set generation for monitors with IBM colour graphics compatible adaptors. DOS 3.0 is fully supported, as are the IBM PC/AT and most of the common PC compatibles.

Keyboard facilities and full-screen editing are improved, and several commands such as)LOAD and)COPY have been substantially speeded up. Graphics displays can now be saved for rapid recall, and the memory scrolling is also faster. Finally there are two major additions to the language which now allows 'ambivalent functions' and the 'equivalence' primitive.

Existing users can upgrade from Release 3 for £95 (plus VAT); the price includes two floppy disks and the necessary extra documentation.

Dyadic Systems Ltd.

In January, Dyadic unveiled a major new release of Dyalog APL, Version 3.0. The event chosen for the announcement was the Uniforum conference and exhibition in Dallas, USA.

Version 3.0 is the result of a major project to optimise performance, and is at least twice as fast as the previous release, with certain operations showing a ten-fold improvement in speed. Several important new enhancements have also been added.

Dyalog APL now follows the same conventions for displaying arrays as IBM's APL2. Simple arrays of two or more dimensions that are too wide to fit on the screen are now 'folded' by row or by page as appropriate, rather than by line as at present. Not only is this neater, but the rules for wrap-around are now consistent throughout. Nested arrays are shown pictorially, as in APL2, in a way designed to show structure as well as contents. Because nested arrays may contain both text and numeric data, this feature can be used to produce simple reports without the need for special formatting primitives.

At Uniforum, Dyadic also demonstrated a pre-release of Cisigraph-C, a new graphics package for Dyalog APL. Cisigraph-C has been developed by Cisi, the Dyalog APL distributor for France. It is derived from Cisigraphe, Cisi's mainframe VSAPL product which is widely used by timesharing customers in France and throughout the world. The new Cisigraph-C takes advantage of Dyalog APL's unique auxiliary processor interface, and follows GKS and SIGGRAPH Core standards. Cisigraph-C is a hybrid system with all its GKS primitives and device drivers written in 'C', while its business graphics functions remain in APL. This gives it the speed of compiled graphics packages coupled with the power and flexibility of APL.

Dyadic is planning a series of seminars beginning in April to introduce Version 3.0 and to demonstrate Cisigraph-C. Anybody who is interested in attending one of these seminars, or who would like a copy of the new Dyalog APL Newsletter, should contact Lesley Gould at Dyadic.

MicroAPL Ltd.

The most important development at MicroAPL is the release of APL for the Sinclair QL. Based on MicroAPL's existing APL.68000 products, QL/APL comes in two versions. These are a new keyword version of the language — specially designed to convert users from BASIC to APL — and a conventional APL character set version. These products cost £99.95 and £129.95 respectively, including VAT. QL/APL has been approved by Sinclair Research for use on the QL.

As well as being a full implementation of the language, with no restrictions, QL/APL offers a number of exciting language enhancements. In addition to such productive features as error trapping, commercial formatter, component filing system, and a very wide range of system functions, QL/APL offers excellent colour screen handling support, direct access to the QL colour graphics facilities, and special features to allow APL use of multiple windows on the screen. QL/APL will run on the basic 128 KB QL, giving around 28 KB of APL workspace, but will also take advantage of extra RAM if this is fitted. This means that workspace sizes of up to 512 KB can be achieved. Execution speed is excellent. being about one-third the speed of MicroAPL Sage and Spectrum systems — which makes QL/APL faster than typical 8088-based micros.

I.P. Sharp Associates Limited

As a part of its continuing expansion plans I.P. Sharp are moving into new and larger offices just off Victoria Street. As from the 1st February the new address will be:

I.P. Sharp Associates Limited
10 Dean Farrar Street
LONDON SW1

Tel: 01-222 7033

The IBM PC XT/370 is now available in this country and runs Sharp APL 25-40 times faster than on an IBM XT. This is certainly fast enough for most applications. Sharp APL on the XT/370 is fully compatible with mainframe Sharp APL allowing code to be developed and run in either environment.

A new 6670 laser printer will be installed in January greatly improving the quality of the high speed print service. Output can be produced directly on A4 with a wide variety of print sizes and fonts.

I.P. Sharp have appointed The Islamun Company as their agents in Bahrain and a new node of I.P. Sharp's communications network IPSANET is being installed. Local dial access should be available in Bahrain in March.

APL*PLUS LIMITED TO TRANSFER TO COCKING & DRURY

Cocking & Drury Limited and STSC Inc, today announced the signing of an Agreement in principle under which STSC would transfer to Cocking & Drury the assets of APL*Plus Limited. The proposed transfer is subject to execution of a definitive Agreement.

APL*Plus Limited has operated since 1979 as a subsidiary of Maryland-based STSC to market and support the parent company's APL-based software products and services. According to John W Myrna, Senior Vice President of STSC's APL products and services business unit, the proposed transfer will align STSC's UK operations with that of its other international operations which are managed through distributors and agencies. "We believe that the customers of APL*Plus Limited will not only retain continuous service, but will realise even greater support in the future with Cocking & Drury," stated Myrna.

Romilly Cocking, Director of Cocking & Drury, said "We are confident that the complimentary strategies of our two companies will strengthen our delivery of APL services to our customers and will enable broader coverage in the microcomputer and mainframe APL software market. We have been dealers for APL*Plus micro products for some while now, and we see the new move as a natural development of our existing relationship."

STSC, an industry leader in the development of the APL programming language, provides APL-based software in the microcomputer and mainframe environments, and publishes micro software written by independent developers through its APL products and services business unit. Other STSC business units include Execucom Systems Corporation and Logistics Decision Systems. STSC has branch offices in major US cities and licenses software through distributors in several countries worldwide.

Cocking & Drury are Europe's leading APL software house. Their services and products include consulting, education, turnkey systems, support and the marketing and supply of software that they or others have written.

MICRO
APL

where APL means business

You know what's good about micros. Fast response; no monthly bills; software designed for users, not programmers; access whenever you like. In a word, independence.

You know what's bad about some micros. No networking; no access to your mainframe data; fewer features than you're used to; no technical and service backup. In a word, isolation.

Wouldn't you like to see a full range of APL micros, from the IBM PC to large multi-user machines? With local area networking, and IBM and ICL terminal emulation? With full lookalikes of features like IBM's AP124 and AP126? All from a company with the commercial and technical muscle to support everything it sells?

MicroAPL Limited
Unit 1F, Nine Elms Industrial Estate
87 Kirtling Street, London SW8 5BP
Telephone: 01-622 0395 Telex: 896885 IOTA

APL PRODUCT GUIDE

Compiled by David Preedy

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.

The second release of the Guide contains several additions of new products and suppliers who slipped through our net at the first pass. 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 on inside back cover.

We reserve the right to edit material supplied for reasons of space or to ensure a fair market coverage.

The listings are not restricted to UK companies and international suppliers are welcome to take advantage of these pages. Where no UK distributor has yet been appointed, the vendor should indicate whether this is imminent or whether approaches for representation by existing companies are welcomed.

For convenience to VECTOR readers, the product list has been divided into the following groups:

- Complete APL Systems (Hardware & Software).
- APL Interpreters
- APL Visual Display Units
- APL character set printers
- APL-based packages
- APL Consultancy
- Other services
- Vendor addresses

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

Note: 'poa' indicates 'price on application'

APL PRODUCT GUIDE

Company	Product	Prices £	Details
COMPLETE APL SYSTEMS			
Alan Pearman	IBM PC & PC/XT	1995 – 4425	Authorised IBM PC dealer supplying complete systems including peripherals.
C.A.C.I.	APL Machine	poa	APL-dedicated array processor from Analogic. Many configurations.
Cocking/Drury	MicroAPL SPECTRUM SAGE II SAGE IV	6000 – 35000	Supplied as part of a turnkey system. See MicroAPL entry.
Gen. Software	Myriade	poa	TI computer with APL and APL operating system
Inner Product	IBM PC	2000 – 6000	IBM PCs supplied for turnkey applications
MBS Rentals	IBM PC & PC/XT	2200 – 5000	For purchase or rental
	COMPAQ	poa	Portable IBM PC compatible
M.B.T.	MBT 10/40	poa	UNIX/68000 based multi-user APL system
	FORTUNE 32:16	poa	FORTUNE 68000 system enhanced for APL
	TORCH	poa	68000/Z80 multiprocessor
MicroAPL	Sinclair QL with QL/APL	from 434	Fully expandable APL system with colour graphics.
	SPECTRUM	11000 – 15000	Expandable multi-user APL computer using Motorola 68000. Standard configuration 1 Mb RAM, 12/36 Mb disc, 12 ports.
	SAGE IV	8500	Multi-user APL computer, 1 Mb RAM, 12/18 Mb disc.
APL INTERPRETERS			
Alan Pearman	APL*PLUS/PC	420 – 695	See APL*PLUS entry
APL*PLUS	APL*PLUS/PC (Release 4)	695	Full feature interpreter for IBM PC, PC/XT, PC/AT & compatibles:CORONA, COMPAQ COLUMBIA,
	Upgrade	95	WANG, OLIVETTI, EAGLE, AJ, ERICSON, ITT

APL PRODUCT GUIDE (continued)

Company	Product	Prices £	Details
APL INTERPRETERS (continued)			
APL*PLUS	APL*PLUS/UNX	poa	Interpreter for Unix systems: WICAT, CADMUS, CALLAN, FORTUNE 32:16, HP9000/500
	APL*PLUS/1000	poa	Enhancements to IBM's VSAPL. Runs under VM/CMS or MVS/TSO.
Burroughs	APL/700	3150	Runs on Burroughs B5000, B6000, B7000 and A9 mainframes.
Cocking/Drury	APL*PLUS/PC	600	Discount on multiple orders.
Dyadic	Dyalog APL	1000 – 8000	2nd generation portable APL for UNIX systems eg. VAX, HP9000, Fortune, MC68000, Zilog, Perkin-Elmer, Perq etc.
Gen. Software	APL*MYRIADE	poa	Runs on Texas Instruments TI990 range.
IBM UK Product Sales	IBM PC APL	poa	With event-handling, & APs for full-screen I/O, disks, diskettes, asynch. comms.
Inner Product	VIZ::APL	250 – 350	8-bit Zilog Z-80 CP/M
	APL*PLUS/PC	600	See APL*PLUS entry
M.B.T.	Dyalog APL	poa	See Dyadic Systems entry
	MBTAPL	poa	Enhanced Dyalog APL for MBT hardware.
	VIZ::APL	poa	Customized for TORCH hardware
MetaTechnics	APL*PLUS/PC	695	Includes utility package free.
	Upgrade kit	95	
MicroAPL	APL.68000	1000 +	Full implementation with component files, error trapping etc. for SPECTRUM, SAGE & other MC68000-based computers.

APL PRODUCT GUIDE (continued)

Company	Product	Prices £	Details
APL INTERPRETERS (continued)			
MicroAPL	QL/APL(keyword)	87	Full keyword APL for QL with many extra features.
	QL/APL(APL chars)	113	VSAPL compatible APL for QL with many extra features.
I.P. Sharp	Sharp APL	25000	For IBM mainframes
	Sharp APL/PC	55	For IBM PC or PC/XT
APL VISUAL DISPLAY UNITS			
Brent-Cybernex	GR1014 Graphics Terminal	1970	Combined APL/ANSI/4010; 1024x1024 res. 2 ports, anti-glare, swivel and tilt.
	APL100	1065	Selectable APL/ASCII; hi-res chars; non-glare, swivel & tilt.
Farnell	Tandberg TDV2221	1498	Ergonomic design APL terminal, 50-19200 baud, 15" anti-reflex screen, low profile keyboard.
	Tandberg TDV2271	1685	Combined APL/ANSI ergonomic terminal as above.
Gen. Software	Mellordata Elite 3045A	400	Second-hand.
Lynwood	Alpha	1995	Full APL character set, 16-bit microprocessor with pixel-addressable monochrome graphics.
	Alpha Colour	2985	As above with 8-colour graphics.
MBS Rentals	Lynwood Alpha	1995	See Lynwood entry.
	Lynwood Alpha Col.	2985	See Lynwood entry.
	Concept/APL	1491 – 1663	See Shandell entry.
	Concept GVT/APL +	2220 – 2393	See Shandell entry
	(All above for purchase or rental)		
M.B.T.	various		Contact MBT for details

APL PRODUCT GUIDE (continued)

Company	Product	Prices £	Details
APL VISUAL DISPLAY UNITS			
MicroAPL	Insight VDT-1	795	Inexpensive APL VDU
	Insight GDT-1	1450	With monochrome graphics
	Tektronix 4105	3750	High resolution colour graphics supporting APL character set on MicroAPL hardware.
Shandell	Concept AVT/APL+	896 – 1195	Conforms to ANSI 3.64 standard; 80/132 columns, 4/8 pages of memory, windowing, 2 comms. ports, low profile keyboard.
	Concept GVT/APL+	1370 – 1825	As above with vector graphics.
Sigmex	Sigmex 5684	from 6091	High-res colour graphics (768x512x4)
	Sigmex 5688	from 7096	High-res colour graphics (768x512x8)
	Sigmex 5472	from 6649	High-res monochrome (1536x1024x2)
	Sigmex 5484	from 10500	High-res colour (1536x1024x4) (Hard-copy output devices available for all these models.)
APL PRINTERS			
Alan Pearman	Epson FX series NEC Spinwriter PRISM 80 PRISM 132		Prices range from 289 to 1995 over all models.
Datatrade	Datasouth DS180	1395	180 cps matrix printer with 2K buffer, 9x7 dot matrix and APL option.
Inner Product	Epson FX80	500	Soft char. set, 160 cps, 80 column
	Anadex 9620	1150	200 cps., 132 col., tractor feed
	Siemens PT88	620	180 cps., 80 col., silent
	TGC Starwriter	1180	40 cps., letter quality

APL PRODUCT GUIDE (continued)

Company	Product	Prices £	Details
APL PRINTERS (continued)			
MBS Rentals	Epson FX series	438-1100	
	Datasouth DS180	1395	See Datatrade entry.
	Phillips GP300	2251	Matrix with letter & draft quality & APL.
	TI745 + APL	1390	Portable dot-matrix.
	DEC LA120	2516	120 cps dot-matrix.
	DEC LA12C	1444	Portable dot-matrix.
	AJ833	2595	Daisy-wheel
	(All above for purchase or rental)		
M.B.T.	Facit 4565	poa	40 cps letter-quality
	Facit 4510/11/12	poa	Matrix printers
MicroAPL	Datasouth DS180	1545	See Datatrade entry
	Phillips GP300	1928	Matrix printer with letter and draft quality and APL.
APL PACKAGES			
APL*PLUS	Info Centre	12000 – 20000	Tools for full-screen entry, display and analysis. Interfaces to other proprietary IC products
	CMCS	30000 – 100000	Complete Materials Management, Factory Planning & Forecasting package.
	Finance/Statistics	275	Comprehensive package for IBM PC and compatibles.
	Tools Vol. 1	poa	Utilities for IBM PC;IRMA 3278 comms, full-screen, RAM disk, report generator
C.A.C.I.	PILGRIM	20000	Language-driven data extraction, and foil preparation. VM/CMS VSAPL, AP126.
	PIM	3000	PF-key driven personal information manager VM/CMS VSAPL, AP124
Cocking/Drury	ARMS 2	3000 – 15000	Applications Generator
	AFM	10000	High performance shared file system.

APL PRODUCT GUIDE (continued)

Company	Product	Prices £	Details
APL PACKAGES			
Cocking/Drury	CALL/AP	3000	Non-APL program execution.
	FORMAT	2250	Enhanced report formatting
	WSAIDS	795	Workspace documentation and development aids.
E&S	PROTOPAK consisting of:		Packages for prototyping management information systems — PC & mainframe
	RMS	550-2500	Relational databases.
	AMS	550-2500	Multi-dimensional arrays.
	RAMS	750-3500	Combined RMS & AMS.
	BMS	750-3500	Dynamic financial modelling & forecasting
	FMS	250	Full-screen handler for IBM PC. (AP124-based)
	CMS	poa	Communications package.
Gen. Software	PROPS	from 500	Spreadsheet system for Product and/or Project Planning.
Holtech	CASH	3500 – 10000	Accounting package and hotel management system on MicroAPL SPECTRUM & SAGE CPUs.
Inner Product	Viewcom	150	Control Viewdata from APL
	APL/DBASE II	150	Interface APL with dBase II
	APL/WORDSTAR	150	Interface APL with Wordstar
	APL/MULTIPLAN	150	Interface APL with spreadsheet
	CEMAS	3500	EEC monetary and agrimonetary analysis.
M.B.T.	RHOMBUS	poa	Integrated Office System
	HASLEMERE	poa	Hotel Accounting System
MetaTechnics	MetaScreen	195	Full-screen handler for APL*PLUS/PC, based on VSAPL AP124

APL PRODUCT GUIDE (continued)

Company	Product	Prices £	Details
APL PACKAGES (continued)			
MetaTechnics	MetaPack	395	Comprehensive utilities package for APL*PLUS/PC. Includes MetaScreen, MetaWS, Browse, Toolbox, Numeric Editor (See advert for full details)
	ADAPTA DLS	poa	Production & purchasing scheduling for process manufacturing.
	ADAPTA MSP	poa	Job-shop loading & scheduling for multi-stage production.
MicroAPL	MicroTASK	250	Project development aids
	MicroFILE	250	File utilities and database
	MicroPLOT	250	Graphics for HP plotters etc.
	MicroLINK	250	General device communications
	MicroEDIT	250	Full screen APL editor
	MicroFORM	250	Full screen forms design
	MicroSPAN	500	Comprehensive APL tutor
	MicroGRID	poa	Ethernet & other networking
	APLCALC	400	APL spreadsheet system
	MicroPLOT/PC	250	For APL*PLUS/PC product
I.P. Sharp	ACT	poa	Actuarial system
	APS	poa	Financial modelling
	BOXJENKINS	poa	Forecasting technique
	CONSOL	poa	Financial Consolidation
	COURSE	poa	APL Instruction
	EASY	poa	Econometric Modelling
	FASTNET	poa	Project Management
	GLOBAL LIMITS	poa	Exposure management for banks
	MABRA	poa	Record maintenance/ reporting
	MAGIC	poa	Time series analysis/ reporting
	MAGICSTORE	poa	N-dimensional database system
	MAILBOX	poa	Electronic Mail
	MICROCOM	poa	Mainframe to micro link
	SAGA	poa	General graphics, most devices
	SIFT	poa	Forecasting system
	SNAP	poa	Project management
	SUPERPLOT	poa	Business graphics
	VIEWPOINT	poa	4GL — Info centre product
	XTABS	poa	Survey Analysis

APL PRODUCT GUIDE (continued)

Company	Product	Prices £	Details
APL CONSULTANCY (prices quoted are per day unless otherwise marked)			
Alan Pearman	Consultancy & Courses	25	Per hour, onsite and offsite
APL People	Consultancy	poa	All levels, most APL systems
Attikale	Consultancy	160	Management reporting systems & Information Centres.
C.A.C.I.	Consultancy	poa	All aspects, specializing in Decision Support Systems.
Cocking/Drury	Consultancy	120-150	Junior consultant
		140-200	Consultant
		185-300	Senior consultant
		275-400	Managing consultant
Delphi	Consultancy	poa	Specialising in management reporting systems and APL on microcomputers.
Dyadic	Consultancy	poa	APL system design, consultancy, programming and training for Dyalog APL, VSAPL, APL*PLUS, IPSA APL etc.
E & S	Consultancy	150-250	System prototyping: all types of information system.
Gen. Software	Consultancy	from 100	
H.M.W.	Consultancy	100-250	System design consultancy, programming.
Inner Product	Consultancy & Training	200	On-site micro/mainframe APL, PC/DOS & Assembler
M.B.T.	Consultancy & Courses	poa	
MetaTechnics	Consultancy	poa	Management Information & Production Management Systems & Process Control.
	MetaSys	poa	Prototyping APL-Hybrid Custom Programming Service.
MicroAPL	Consultancy	150 – 250	Technical & applications consultancy.

APL PRODUCT GUIDE (continued)

Company	Product	Prices £	Details
APL CONSULTANCY (continued)			
QB On-Line	Consultancy	200-250	Complete APL systems.
I.P. Sharp	Consultancy	poa	Consultancy & support service world-wide.
OTHER PRODUCTS			
APL People	Employment Agency	poa	Permanent employees placed at all levels. Contractors supplied for short or long term projects, supervised.
I.P. Sharp	Productivity Tools	poa	Utilities for systems, operations, administration & analysts; auxiliary processors, comms. software, international network.
	Databases	poa	Financial, aviation, energy and socioeconomic.

APL PRODUCT GUIDE (continued)

VENDOR ADDRESSES

Company	Contact	Address & Telephone
Alan Pearman Ltd.	Alan Pearman Maria Pearman	96-98 Chester Road, Huntington, Chester CH3 6BT. 0244-46024
APL People	Valerie Lusmore	3 Cross Lane Close, Orwell, Royston, Herts SG8 5QW. 04626 77375
Attikale Ltd.	Peter Hurdley	51 Brookside, Paulton, Bristol BS18 5YR. 0761-415427
Brent-Cybernex Ltd.	Ken Rumsie	Sovereign House, Dallow Road, Luton, LU1 1TP. 0582-452020
Burroughs Machines Ltd.	M.J. Fennel	Heathrow House, Bath Road, Hounslow, Middlesex TW5 9QL 01-750 1400
CACI Inc. International	Paul Thornett	Oriel House, 26 The Quadrant, Richmond, Surrey, TW9 1DL. 01-940 3606
Cocking & Drury Ltd.	Romilly Cocking	16 Berkeley Street, London W1X 5AE. 01-493 6172
Datatrade Ltd.	Tony Checksfield	38 Billing Road, Northampton, NN1 5DQ. 0604-22289
Delphi Consultation Ltd.	David Crossley	Church Green House, Stanford-in-the-Vale, Oxon SN7 8LQ. 03677-384
Dyadic Systems Ltd.	Peter Donnelly	30 Camp Road, Farnborough, Hants. GU14 6EW. 0252-547222
E & S Associates	Frank Evans	19 Homesdale Road, Orpington, Kent BR5 1JS. 0689-24741
Farnell International Instruments Ltd.	R. Fairbairn	Jubilee House, Sandbeck Way, Wetherby, W. Yorks. 0937-61961
	or Roger Attard	Davenport House, Bowers Way, Harpenden, Herts. 05827-69071
General Software Ltd.	M.E. Martin	22 Russell Road, Northolt, Middlesex UB5 4QS. 01-864 9537
H.M.W. Programming Consultants Ltd.	Stan Wilkinson	142 Feltham Hill Road, Ashford, Middlesex TW15 1HN. 07842-41232

APL PRODUCT GUIDE (continued)

VENDOR ADDRESSES (continued)

Company	Contact	Address & Telephone
Holtech Ltd.	Jan Bateman	'O' Block 4th Floor, Metropolitan Wharf, Wapping Wall, London E1 9SS. 01-481 3207
IBM UK Product Sales Ltd	Stephen Voller	PO Box 32, Alencon Link, Basingstoke, Hants., RG21 1EJ. 0256-56144
Inner Product Ltd.	James Manning	Eagle House, 73 Clapham Common Southside, London SW4 9DG. 01-673 3354
Lynwood Scientific Developments Ltd.	Gareth Wokes	Park House, The High Street, Alton, Hants GU34 1EN. 0420-87024
M.B.S. Rentals	Robert Notley	119/120 High Street, Eton, Windsor, Berkshire. 07535-68171
MetaTechnics Systems Ltd.	Steve Margolis Dave Toop John Stembridge	Unit 216, 62 Tritton Road, London SE21 8DE. 01-670 7959
MicroAPL Ltd.	Richard Nabavi	Unit 1F, Nine Elms Industrial Estate, 87 Kirtling Street, London SW8 5BP. 01-622 0395
Modern Business Technology Ltd. (M.B.T.)	Michael Branson	P.O. Box 87, Guildford, Surrey GU4 8BB. 04868-23956
QB On-Line Systems	Philip Bulmer	5 Surrey House, Portsmouth Road, Camberley, Surrey, GU15 1LB 0276-20789
Shandell Systems Ltd.	Maurice Shanahan	12 High Street, Chalfont St. Giles, Bucks HP8 4QA. 02407-2027
I.P. Sharp Associates Ltd.	David Weatherby	10 Dean Farrar Street, London SW1 01-222 7033
Sigmex Ltd.	Alan Taylor	Sigma House, North Heath Lane, Horsham, W.Sussex, RH12 4UZ. 0403-50445

BOOK REVIEWS

Compiled by Robert Bittlestone

Books are welcomed for review on any topic likely to be of interest to the APL user, and should be sent to the editor on the address on the inside back cover.

Learning and Applying APL
by B. Legrand
Published by John Wiley and Sons (1984)

Reviewed by Anthony Comacho

This book is a disappointment, not because of the technical content or the way APL is explained but because of the bad translation and the poor job of book production.

First I will deal with these disadvantages and then for anyone who is interested, describe M. Legrand's approach to teaching APL - a review of the book the author wanted!

Translation

The translator was Julian Glyn Matthews of Linguatech. In spite of his English-sounding name he is not a native speaker of English. It is very rare to get an idiomatic translation from anyone not native in the target language and the translator should also be familiar with the usage of the subject's technical terms in the target language. You may judge how far Mr. Matthews falls short: instead of 'To pick out the value 8520 from the array TYRES' he writes "In order to designate the value 8520 to the array TYRES" (section 2.13). In another place he writes "If it is not followed by a value the branch arrow (→) causes the output of a function and all the functions which call it, which the instruction (→0) does not allow" (section 4.36). Perhaps you would agree with me that you can only disentangle this if you already know what he is trying to say.

These examples are striking but they are not alone. Mistranslations are common: I seem to have found a dozen or so in each chapter. To discern the author's meaning I often had to call on my rusty French and my APL experience. The translation from French is so literal that one can often guess at the original and get at the meaning through a retranslation. Take the rule on page 17 for example:

> When a vector is indexed by an array, an array of the same dimensions as the index is obtained, in which each value would have been replaced by the element of the vector which it designates.

The phrase "would have been" may be right in French; in English "is" or "has been" is more natural. As it is, the rule gives the feeling that the value wasn't replaced because "would have" leads one to expect an "if" followed by the condition under which the replacement "would have" been made.

The translator clearly does not have experience of using APL in English: he writes that a function *restores* its result (instead of 'returns'), he calls the pi-times symbol *orange* and the del symbol *carrot*, he refers to the *line* operator (instead of 'axis') and to the quad symbol as 'window'. Where the French may be precise, this translation is not as it loses the case and gender association between noun and pronoun — "when binary values are available there are logical functions which enable them to be composed from them."

It may be amusing at first to find "the computer *passes* a blank line between each sub-array" (instead of 'prints' or 'outputs'); it may seem fun to calculate *mobile* averages, print subtotals at *ruptures*, preserve your workspace by *safeguarding* it and observe the *processings* of your functions: it soon palls. The French for 'byte' is apparently 'octet'. The translator has left it as octet throughout.

Book Production

The book is produced in typescript from a daisywheel typewriter. It uses the typewheel Courier for the text, Orator for headings and Light Italic for page headers and APL, except for a few places where a different printer seems to have been used.

I have used a daisywheel printer with a word processor to produce a book and it is an excellent method. The company that prints the book can't add mistakes if it is given camera-ready copy. To be clear APL lines need consistent alignment so that the six space prompt is easy to recognise.In this book the alignment is not quite consistent and someone has used rub-on shading to distinguish keyboard entries. The shading is not skilfully applied and makes many pages look a mess.

There are many corrections to this text that are misaligned as if the correction was made with paint-out liquid and then clumsily overtyped. It would have been much better to use a word processor and reprint the corrected pages. There are still many typing errors. There are *principle* arguments, one thing has *lead* to another and *preceeded* a third - the latter so often that I suspect the translator. Other errors are surely misprints, such as 2 for iota, stile or even slash for backslash, capital I for I-beam, capital V for del and a monadic header as an example of a dyadic function. A newcomer to APL would find examples with such errors inexplicable. I estimate that there is on average one error per page. There are misprints even in the page headers. Attempts have been made to retouch some pages manually - erasures and alterations in ink. Where tables had to be ruled they are sometimes not even ruled straight (see section 4.22).

There are other inconsistencies too. In chapter 4, for example, the text repeatedly refers to a function INCREASE whereas the examples given are called BIGRISE. Similarly there is a function CONSULT sometimes called CONSULTATION in the text (it is even listed as that in what purports to be the computer response to ')FNS '). In section 2.6 an array is defined generally so that a scalar is an example and then restrictively so as to exclude scalars. Section 1.6 says "You can now start reading this book....." In section 5.4 we are told that a bar chart is often incorrectly called a histogram: the function given as an example, which draws bar charts, is called HISTO!

Approach to APL

The book begins at chapter 2 and after two pages introduces a rank 3 array called TYRES. This daring is repeated elsewhere. There are surprises and tricks that kept me awake and interested. Every chapter ends with a collection of exercises, and these are one of the best features of the book. If you teach APL and want examples there is a seam here to be mined.

The attempt to be comprehensive is worthy but sometimes inadequate, particularly at the end of the book. I suspect the author cannot speak with authority of all the APL implementations. Here is a complete paragraph:

> Some systems enable special characters to be created by overstriking two letters. For example 'E' which is L and F overstruck, gives the character Line Feed. The reading of functions is complicated by this.

My appetite was whetted. I wanted to know which systems do this and to understand the problem and whether there are other such oddities: M. Legrand leaves me unsatisfied.

Chapter 3 explains some of the primitive functions to give material for the exercises and examples of defined functions in chapter 4. I liked the way M. Legrand introduces ravel, take and drop by setting them as exercises to be written as defined functions using only previously explained primitive functions.

Chapter 5 introduces operators thus:

> Operator is the name given to a function when one at least of the arguments is a function or another operator. Hence the notation +/A indicates that the operator / applies to the function +. The whole +/ comprises the RESULTANT FUNCTION, the operand of which is A.

Some people may find this insufficiently rigorous, but I doubt whether anyone would be really confused by it. I could have complained that he defines an operator as a function, but I think in this case it would be unfair. There are two places where similar ways of explaining things are used, which cause no confusion in the classroom but which need more careful wording to exclude misunderstandings in a book.

Chapter six deals with operations on arrays, chapters seven, eight and nine with the remaining primitive functions, ending with a comparison between a brief introduction to quad-FMT and 'format'. Chapter 10 is called INTERFACES WITH THE COMPUTER, SHARED VARIABLES and is the first place where workspaces, the index origin, the random link, the comparison tolerance and the latent expression are dealt with. Chapter 11 is about files (mostly STSC component files). Chapter 12 is about function development (mostly the del editor and the stop and trace facilities). Both these chapters are rather sketchy; neither of them covers the range of filing or editing features now available.

Chapter 13 is entitled STYLES, METHODS AND ADVICE. I agree with most of the advice given but dislike the didactic tone and the absence or weakness of the reasons for it. I feel the lack of a frame of reference within which conflicting requirements such as brevity and comprehensibility could be balanced. Readers will also be puzzled to know what action they should take to follow advice such as:
Care should be taken that the performances obtained cannot be brought into question by a change of equipment.
This last chapter also suffers badly from the inadequacy of the translation.

M. Legrand clearly is an optimist about the chances of APL becoming the language of the 90s:
In 1990, programs written in FORTRAN, COBOL, and PL/1, will still be used, but new developments will occur essentially with the aid of APL, or totally in APL.
I hope he is right but have my doubts.

Build your own Expert Systems
by Chris Naylor
Sigma Technical Press (1983)

Reviewed by Robert Bittlestone

The volume of literature on expert systems is mushrooming at present, but most of the publications tend to have titles like "Fuzzy dynamic programming under conditions of non-trivial heteroscedasticity; a critical reappraisal" with authors such as Baas & Kwakernaak. It is consequently a matter for some encouragement that with this book we are faced with the high probability of being able to pronounce not only the title but also the author's name correctly.

Naylor is a well-known populist in the computer world and a frequent contributor to the technical press. The book is subtitled "Artificial intelligence for the aspiring microcomputer - with listings for Apple II and Sinclair Spectrum", so we know where we are before opening the cover. The contents themselves consist mainly of a comprehensively worked example of an expert system written in BASIC, which delves into necessarily labyrinthine complexity in its attempts to handle topics such as recursion and local variables, thereby unwittingly becoming perhaps the best indirect advertisement for APL one has ever seen.

Naylor does most things very well indeed; in particular his descriptions of MYCIN, PUFF, DENDRAL, and PROSPECTOR in Chapter 9 are obviously researched from the original material rather than hastily regurgitated from someone else's article. His examples all seem to work, at at least the conceptual level. If you share a background in APL the task of entering all this code in BASIC is however superhuman, particularly if your only recollection of APPLE BASIC is knowledge of the vital "3 DOG" operating system restore kluge, which was to the best of my belief first discovered by David Eastwood in 1979.

Inevitably this leaves one or two things done badly. The first problem for many readers is going to be Naylor's chummy style of prose. I am mindful of exhibiting most of his own worst habits in this very review, but in my defence I would argue that a book review is brief and it is the reviewer's task to keep his audience awake. Naylor seems to feel it necessary to present himself as a cross between an agony column and a computerised weather forecast. Perhaps, however, this is psychologically desirable if you program in BASIC over any length of time. The second problem is that all this mateyness dispels the possibility of a sense of awe at some of the more impressive accomplishments in this field. This, I think, is a pity, particularly bearing in mind the average age of the target population. But it's easy to criticise and harder to contribute; all in all a rattling good yarn and highly recommended to the younger reader.

Expert Systems: Concepts and Examples
by Alty and Coombs
NCC Publications (1984)

Reviewed by Robert Bittlestone

Don't buy this book unless you are seriously interested in finding out about expert systems, and fairly seriously interested in creating one for yourself. However, such aspirations are a sufficient as well as necessary reason for purchase; here in one modest title is really everything you need to launch yourself into this popular new activity. Part I is a fast brush-up on concepts and techniques. This time we manage to avoid BASIC in favour of LISP, ((although whether) (this is) (an unqualified advantage) (remains to be seen) (.))

Part II is a really thorough tour through MYCIN, PROSPECTOR, INTERNIST, DENDRAL, R1 and MOLGEN. If some of these names are starting to sound familiar, it's because there are in fact only about a dozen expert systems of sufficient grandeur that everyone wants to write about anyway, so the same old names keep coming up. This makes it correspondingly easy for the neophyte to namedrop. The one I like best is R1, which DEC now call XCON. This is an expert system which allows salespersons and others of vanishingly low intelligence to order DEC VAX computers. For example:

```
IF: The most current active context is assigning a power supply
    And an SBI module of any type has been put in a cabinet
    And the position it occupies in the cabinet (its Nexus) is known
```

```
       And there is space available for a power supply for that Nexus
       And there is an available power supply

THEN:  Put the power supply in the cabinet in the available space
```

One feels an overwhelming compulsion to add to the knowledge base the rule:

```
ELSE:  The bloody thing won't work.
```

Rather more relevant, surely, is the metarule:

```
  IF:  The VAX machine is this complicated
       And your salesmen are too dumb to order it properly
       And you have a whole lot of spare research cash for some reason

THEN:  Spend the cash simplifying the machine or hiring new salesmen.
```

However, none of this is the fault of Alty or Coombs and the book remains a bargain for the aficionado.

ARTIFICIAL INTELLIGENCE: Tools, Techniques and Applications; O'Shea & Eisenstadt, Harper & Row 1984.
Available from Harper & Row, Tavistock Street, London WC2.

by Robert Bittlestone

This is a collection of fifteen papers by well known innovators in the AI field. When reviewing an anthology one may criticise either the authors for infelicities in their papers, or the editors for lack of balance in their selection, or indeed both. Equally one may instead praise all concerned, my own approach in these particular circumstances. This is an invaluable source of otherwise hard to locate contributions.

Under "Tools" we get the by now mandatory introduction to LISP and PROLOG, with an interesting variant on POPLOG from the Sussex University team. If this shows that a new language can be presented as acceptable for expert systems, what about APL? In the Techniques section the emphasis is on robotics, with an amusing addition called "How to get a Ph.D in AI". Finally, the Applications heading includes various case studies, some of them unpublished elsewhere as far as I know, ending with the inevitable walk through MYCIN.

IBM PC Expansion & Software Guide; QUE Corporation.
IBM PC & XT Assembly Language: A Guide for Programmers; Scanlon, Prentice-Hall 1983.
Graphics Primer for the IBM PC; Waite & Morgan, Osborne(McGraw Hill).

Reviewed by Robert Bittlestone

Here are three books about the IBM Personal Computer, all available from Computer Bookshops Ltd., Lincoln Road, Olton, B27 6PA, who kindly offered them for review.

The Expansion and Software Guide is perhaps the most invaluable and complete work of its kind. The publishers are to be congratulated for compiling a detailed encyclopaedia of who makes what hardware and software for the IBM PC. Evidently — perhaps somewhat shamefully — almost all of the entries concern US companies, and it would be useful to have a UK appendix in which the details of local distributors were listed. However, there's no minimum charge for international direct dial to America and if you can cope with the 8-hour time lag (most of the suppliers are in California) a brief call secures such information. Recommended for any serious user of the IBM PC.

Many books have now appeared about IBM PC assembler programming and I am certainly not qualified to make a comparative review. For me, assembler is just like a car engine. The theory always sounds very simple, but when you open the bonnet it takes half an hour to distinguish the starter from the generator. After an hour of reading about registers I usually feel sufficiently daring to consider transferring the contents of register A to register B, but inverting a matrix sounds a bit harder. However, Scanlon's book is probably as sympathetic as it is possible to be for a dolt of my lack of calibre and I recommend it to all like-befuddled would-be machine codists.

The book on graphics I enjoyed, partly because graphics are easier to picture than indirectly addressed overhead camshaft hex offsets, and partly because the examples assume programming in BASIC, which even I can understand. IBM's own manuals on their Colour Graphics Adaptor are hardly a masterpiece of clarity and this volume is much more sympathetic. If you read through it you will certainly understand the bewildering number of different modes in which the colour card works. However, despite the clever things that can be done with the hardware, I suspect you'll also end up agreeing how imaginative IBM have been in restricting the user to 320 by 200 pixels for normal colour display. This imposes a limit on resolution which is unacceptable for most serious business purposes. However, this is hardly the fault of the book, which copes manfully with the constraints of its subject.

SOFTWARE REVIEW
U-REG: (User-friendly Regression)

by Teresa Jones

This is a regression package supplied by UFSS who are associated with the Applied Statistics Research Unit at Kent University. It is written in APL and can be run on a mainframe or a PC with an APL interpreter.

The package is easy to use - one can merely type *)LOAD UREG* and the package itself will tell you how to proceed, with extra help if required in answer to ?, HELP, or SOS. The standard prompts can be cut to a bare minimum as one becomes used to them, and answers to questions can be stacked if you know what is coming next.

Data can be input manually, from an APL matrix stored elsewhere, or from one of UREG's own rather idiosyncratic data files. This is reasonably flexible, although if you already have a standard filing system it would be nice to use that instead.

Helpful

From a statistical point of view the package is very helpful. You are given ideas and hints on sensible things to try, such as the suggestion that you should visually explore your data to check for unlikely values. It also reports such things as sequential ordering of data and possible qualitative observations.

The package then leads you on through different stages (which can be bypassed using NEXT) allowing you to transform your data if necessary. It then calculates a regression equation on those variables you wish to include; all the usual statistics are available. If your equation did not give a particularly good fit UREG may suggest that you transform one of the variables, for example to use the log of it, and try again. On completion all the data is automatically saved.

Presentation

The presentation of the results is not quite as good as the content - as our VSPC system has no 'offline' facility we are limited to simple screen hardcopies, or a printed log of the whole terminal session. There are also some problems when many columns of data are being used; the results are not neatly formatted, but wrap round making it difficult to extract the required information.

The amount of data is restricted by the system you run on. To get results on a 'large' (530 observations on 37 variables) data matrix around 700K of workspace is required to avoid WS FULL problems.

Summary

On the whole we are pleased with the package despite its idiosyncrasies, and we hope to make good use of it. It is possible that some of the functions could be tailored to suit us, and UFSS seem fairly happy for us to do this. Of course we may then encounter problems as new releases of the software arrive! The package is very usable in its own right as long as you don't expect it to slot in too neatly with everything else at your installation.

MetaPack... EVERYTHING YOU NEED FOR APL ON THE PC

MetaPack is the comprehensive utilities package for use with STSC's excellent APL* PLUS/PC. Many of the utilities are written in Assembler for efficiency.

MetaPack comprises:

MetaScreen — the popular AP124 based full screen handler.
NI — full screen editing of APL numeric variables.
Browse — vertical and horizontal scrolling of reports or text files.
Tool Box — an invaluable collection of APL utilities.
MetaWS — program filing system for easy management and documentation of large APL systems.

MetaPack **£395***
MetaScreen available separately **£195***

● Enhanced release 4 of **APL* PLUS/PC** now available ●
Upgrade kit for release 3 .... **£95***

Special Offers

APL* PLUS/PC release 4 + **MetaPack** **£895***
APL* PLUS/PC release 4 + **MetaScreen** **£695***

APL*PLUS/PC TOOLS volume 1 .... **£295***
Utilities for communication, screen management, output formatting, disk and file management and software development.

APL/ PLUS/PC Financial & Statistical Library **£275***
Comprehensive library of financial, forecasting and statistical routines.

For IBM PC XT AT, COMPAQ & Compatibles.

Full consultancy and support service. ***all prices ex VAT.**

MTS MetaTechnics Systems Ltd
Unit 216, 62 Tritton Road, London SE21 8DE, 01-670 7959

The Systems Builders

RECENT MEETINGS

This section of VECTOR is intended to document the seminars delivered at recent meetings of the Association, particularly for those members who work outside London and often find it hard to spare the time to attend.

We are dependent on speakers for their willingness to provide us with a written version of their seminars, and we would remind them that "a picture's worth 1000 words". Copies of slides and transparencies will enhance their articles.

The Activities officer (details on inside back cover) will respond enthusiastically to offers from individuals to contribute seminars and supporting papers.

Introductory Notes

by Adrian Smith

In this issue we cover three out of the four Autumn meetings of the Association, plus the 1984 I.P. Sharp Users Meeting. Papers and discussion from the November seminar 'Information Centres Revisited' are being held over to the next issue; this is partly to keep this one down to a reasonable size and partly because the change of venue has rather thinned out the spring programme.

'APL and Knowledge-based Systems' was held on September 21st, and has yielded two excellent papers by M.J. Winfield and Christopher Harvey. The second of these is really a case study of a 'Do it Yourself' expert system, so it is to be found under the 'Case Studies' heading at the end of the general papers.

'How to Survive in XXAPL' was a workshop held on October 19th. The collection of notes on VS APL under CMS shows how a little work can add a great deal of power to this particular flavour of APL.

'What's New for 1985' was an ambitious day-long sequence of product forums and exhibitions. Dick Bowman gives his reflections as organiser, and I have included some of my own comments on some of the more exciting announcements. Many thanks to Roy Tallis for acting as photographer for the day.

'IPSA 1984 Users Meeting' was held in Toronto in October. This was not at all a parochial event, and I hope my notes will be of interest to all APL users, not just those on the Sharp system.

EXPERT SYSTEMS - What are they and who needs them?

by M.J. Winfield

Introduction

Expert Systems were first devised in the mid 1960's by the Artificial Intelligence (A.I.) community and may be considered as one of the areas of applied A.I. More specifically they are a sub-group of the more general class of Knowledge Based System (K.B.S.).

The use of these expert systems has opened up radically new application disciplines for using the computer. They are problem solving programs that solve problems generally considered to be difficult and requiring expertise, and that will operate at the expert's level. Unfortunately many of the Expert Systems are very domain specific but this is likely to change over the coming years.

What is an Expert System?

Before we can define what an Expert System is, we need to look at the broader set of K.B.S. A K.B.S. is a system which represents 'knowledge' in a formal scheme so that a computer can manipulate it to perform a task or tasks. An important point to note is that the 'knowledge' is a quite separate entity from either the program or any input data. The knowledge is symbolic data which is highly structured and is used to model the relationships between the data items and the way they can be used. There are many different data structures which are used for representing knowledge in a computer. The most common method in use at the moment is 'Production Rules', but more about knowledge representation later.

As we said earlier, Expert Systems are a sub-group of the more general class of K.B.S. This new technology is being used at the present time to construct fairly simple Human-Computer Systems which, none the less, work within a particular domain, at a level of competence comparable or better than a human expert. Considerable research is being undertaken to investigate methods and techniques for building more complex Human-Computer Systems.

Expert Systems are expected to behave as intelligent systems; that is they are said to perform reasoning by matching and manipulating symbols, grasp the fundamental principles of the application domain, solve problems which we consider to be complex and interact with the user in an intelligent way. It is now possible to look at a formal definition of an Expert System.

An Expert System is a computing system which uses knowledge of a problem domain in such a way as to enable it to perform as a skilful and cost-effective consultant and to produce, on demand, explanations of the reasoning processes it used.

How Do We Recognise an Expert System from a Conventional Non-Expert One?

For a system to be considered expert it must perform a task which requires the knowledge and skill of an expert in order to perform it; for example to perform an analysis on the potential of a site for finding ore deposits requires the skill of an expert geologist whereas the solution of large matrices is mundane and can be performed by rote rather than needing expert knowledge. An Expert System should have a good user interface, allowing communication to be natural for the user. The format and language used should be one which the user is familiar with and happy to be using with the problem domain; this means that a full natural language interpreter will not be available, but communication will more likely take place via a problem domain oriented language. The system, on demand, should also be capable of giving explanations of the reasoning processes it used.

Constituent Parts of an Expert System

Although all Expert Systems vary in the actual way thay are constructed, they all use a number of generic parts. The differences are really associated with the way the parts are built. The way the individual parts are linked together to form an Expert System is shown in Figure 1.

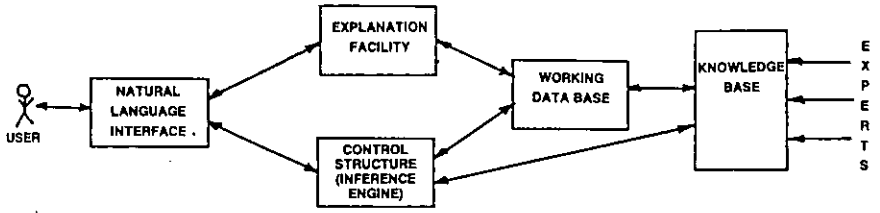

USER
NATURAL LANGUAGE INTERFACE
EXPLANATION FACILITY
CONTROL STRUCTURE (INFERENCE ENGINE)
WORKING DATA BASE
KNOWLEDGE BASE
EXPERTS

Figure 1

It is worth examining each of these parts separately in order to understand their function.

Knowledge Base

This is where the domain specific knowledge is stored in symbolic form, according to some data structure. The knowledge takes the form of facts about the task domain and heuristics that guide the system in the use of the knowledge for solving problems within the domain. The knowledge base is built by using the knowledge gleaned from a domain expert. In most Expert Systems a uniform representation is used for the knowledge but it would appear that as the knowledge base becomes larger this may lead to a problem regarding efficiency. Consequently increasing attention is being given to the use of non-uniform representations for increasingly large knowledge bases.

Working Data Base (Blackboard, STM)

Most problems cannot be solved in one step. Consequently the way an Expert System solves a particular problem is to break it down into a number of smaller problems. The important point to remember about an Expert System is that it does not have to be told the steps that the problem has to go through in order to solve it. The system will work out the sequence for itself. The working data base is used to hold the current status of the problem being solved. That is to say it should maintain a record of all the data items supplied by the user and those the system has inferred for itself. All Expert Systems use some type of intermediate decision representation although its actual structure will vary from system to system.

The Inference Engine

The inference engine is an interpreter which has two very important features, a powerful pattern matcher and a control regime. The inference engine drives its way through the knowledge base until it finds a data structure which matches with the data in the working data base, i.e. matches with the current step in the problem situation. Once a match is found the data structure is fired or instantiated and used for drawing inferences for the next stage in the problem situation. The control regime determines how the interpreter looks for the data structures (i.e. a search problem), and how any fired data structures should be handled (i.e. irrevocably or tentatively). (See Ref 2 for a good exposition of control regimes.)

When an irrevocable control regime is used, the knowledge from the fired data structure is applied and once it has been applied there is no way of returning to the original state. This is not normally an acceptable situation and consequently it is usual for Expert Systems to use a tentative method. Such a scheme enables the data structure to fire, but if at a later date it is realised that the line of reasoning being pursued is false or not leading anywhere, then it is possible to revert to an earlier state by back-tracking up the solution tree.

The search problem is common to most A.I. programs, and is one which is well understood and documented. Search in Expert Systems is generally accomplished by using a depth first method with back-tracking around an AND/OR tree which is generated as the problem solution is developed.

If a rule based Expert System is used then it may run as a forward driven system, a backward driven system or a combination of both. A forward driven system means that the rules are fired by matching the attributes on the 'situation' or 'IF' side of the rule. (See Figure 2a). Once all of the attributes of a rule are satisfied, that rule will fire, and the 'actions' or 'THEN' side of the rule will be performed. However if a system is running as a backward driven one, then a hypothesis

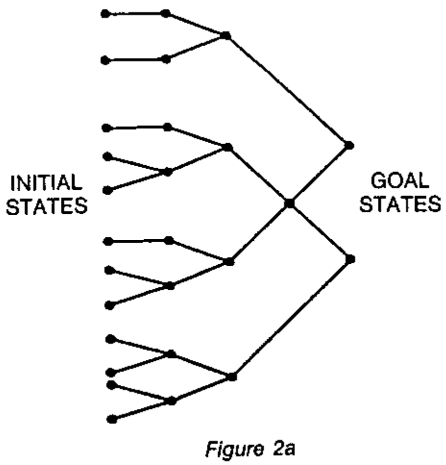


Figure 2a

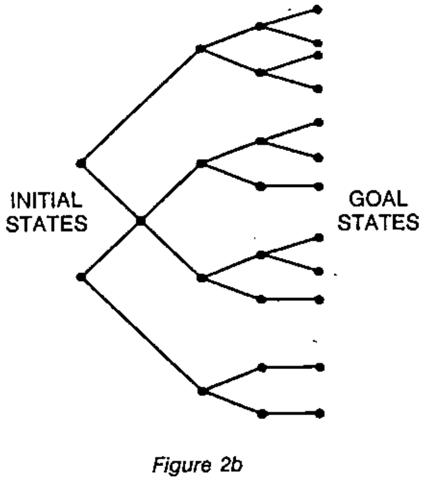


Figure 2b

is put forward and a search is made of the 'action' side of the rules to find a rule(s) which matches the hypothesis (see Figure 2b). Any rules established as being candidates are looked at and an attempt is made to prove the hypothesis by satisfying the 'situation' side of the rule.

Natural Language Interface

In order to allow the user and Expert System to communicate with each other an interface is needed. Although it would be nice to have a full-blown natural language interface this is not possible with the current state of the art. Communication generally takes place using a problem-oriented language, usually a restricted version of English. The interface should take care of parsing and interpreting users' questions and commands, as well as formatting information generated by the system including explanations and answers to questions.

Explanation Facility

The explanation facility is used to give, on demand, an explanation of the reasoning used by the system in solving a particular problem. This normally requires the sequence of knowledge items used to be recorded. In many current Expert Systems a tree structure is built as the problem solution unfolds, and this is then used during the explanation process. Because Expert Systems are working in complex domains it is important for them to possess an explanation capability, since it will:

- help the expert and knowledge engineer to refine and improve the system;
- give the user more confidence in the recommendations or problem solutions given by the system.

It is worth remembering that explanations given by most human experts are generally tailored to the level of their audience; that is, a much more simplistic view can be used for naive users than for expert users. To do this requires a level of intelligence far greater than that possessed by any current Expert System.

KNOWLEDGE REPRESENTATION

As we have already mentioned Expert Systems are very dependent upon the knowledge base if they are to operate at an expert level. Such systems, unlike conventional data base systems, require a knowledge base with divers kinds of knowledge. These include, but are certainly not limited to, knowledge about objects, processes and very hard to represent commonsense knowledge about goals and actions. How this knowledge is represented in a knowledge base is an important consideration for any Expert Systems builder. The three most popular methods used in Expert Systems are Production Rules, Logic and Frames. It is worth looking at each of these very briefly.

Production Systems

The idea of a Production System as we know it today was proposed by A. Newell (9) as a model of human reasoning, although the original idea of production systems was first proposed in the 1940's. The knowledge is represented as Production Rules which consist of two parts, a situation recognition part (i.e. a pattern) and an action part, see Figure 3 for an example of a rule from the Expert System XCON.

SITUATION	ACTION
(List of things to look for - pattern)	(List of things to do)

The rule will only fire, that is the actions will only be performed, when a complete match is found to all conditions on the situation side. The conditions are matched from the items of information held in the working data base. Once a rule has fired, the data base is updated and the cycle repeated until a goal is found.

```
VERIFY-SBI-AND-MB-DEVICE-ADEQUACY-3

IF:      THE MOST CURRENT ACTIVE CONTEXT IS VERIFYING SBI AND
         MASSBUS DEVICE ADEQUACY
AND      THERE ARE MORE THAN TWO MEMORY CONTROLLERS ON THE
         ORDER

THEN:    MARK THE EXTRA CONTROLLERS AS UNSUPPORTED (i.e. NOT TO
         BE CONFIGURED).
AND      MAKE A NOTE TO THE SALES PERSON THAT ONLY TWO MEMORY
         CONTROLLERS ARE PERMITTED PER SYSTEM
```

Figure 3

This method has proved a very natural way for both extracting and encoding knowledge in many Expert System applications.

Logic Systems

First-order logic started to show its usefulness for representing knowledge in the 1960's, as a result of research in mechanical theorem proving. Since then research has looked at logic formalisms with a view to putting them into a more conventionally oriented framework, particularly via PROLOG where all knowledge is reduced to a collection of Horn clauses.

Frame Systems

Frames were first postulated by Minsky (10) as being a way of handling stereotyped situations. Much of their power relies on the inclusion of expectations and other kinds of prescriptions.

A frame represents an entity and contains a number of slots. Each slot will have a name and may be empty, contain things relevant to the frame entity, or specify a further frame (Figure 4). It is possible for slots to have default values which will automatically be used when the frame is instantiated, unless new items are known which fit the situations better, in which case the default values can be overridden.

Frames are commonly used in a number of applications like computer vision, natural language understanding and speech recognition.

They have also been used in Expert Systems, one in particular being NUDGE (11) which is used for scheduling events.

This system takes incomplete requests for events to be scheduled and transforms them into more complete requests by filling in any missing information which humans often take for granted.

Although we have very briefly considered three methods of representing knowledge there are many more, some of which relate to the above methods, others which do not. Knowledge representation is an increasingly important part of A.I. and has a very active research community. For a good resume on knowledge representation, readers are referred to (12).

REASONING WITH UNCERTAINTY

Under many circumstances the expert is expected to come to a decision or offer advice when the information about the problem situation is uncertain or incomplete. If experts are expected to work under these constraints then Expert Systems, working in similar situations, should also be expected to do so.

One of the earliest Expert Systems to perform reasoning with uncertainty was MYCIN, which was expected to represent judgmental reasoning of the type 'A suggests B' and 'D and E tend to F'. Numbers, known as certainty factors, were used to indicate the strength of a rule. Using these certainty factors, MYCIN allows evidence confirming the hypothesis to be collected separately from that disconfirming the hypothesis. The 'truth' of a hypothesis, at any particular time is the algebraic sum of the evidence confirming it. When the premise of a rule is evaluated, each predicate returns a number in the range -1 to 1. MYCIN works on the basis that an 'AND' performs minimisation while 'OR' performs maximisation of the arguments. Providing the result's premise value is greater than an empirical value of 0.2, then the conclusion is presented with a certainty of:

Premise value * Certainty Factor (13)

Although the MYCIN certainty factors were derived from probabilities there are distinct differences. Some criticism has been levelled at the MYCIN method, arguing that it is an ad hoc approach and could have been replaced by a more thoroughly studied approach based upon Bayes Rule. Some Expert Systems, particularly PROSPECTOR, use Bayes Rule for calculating uncertainty. The major problem with this approach is the large amount of data needed to determine the conditional probabilities which are used in the formula. Often conditional independence is assumed, owing to the very large quantities of data needed for determining the independence. This has led to arguments that such assumptions on independence undermine the rigorous statistical model.

An approach which is being looked at in some detail is the idea of fuzzy logic as discussed by Zadeh (14) and others. Using fuzzy logic, a statement like 'X is a tall building' is interpreted as having an imprecise denotation which may be characterised by a fuzzy set. This is shown as:

Fuzzy Proposition	'X is a tall building'.
Fuzzy set	(X (20,40), 0.1) (X (40,100), 0.3) ({X > 100}, 0.7)

Fuzzy logic details the laws of inference for fuzzy sets. We can expect to hear much more about fuzzy systems in the future.

KNOWLEDGE ENGINEERING

One of the major tasks in constructing an Expert System is the implementation of the knowledge base. It is now recognised that the power of an Expert System comes from the knowledge it has, not its inference engine. Therefore the collection of the knowledge and the structure of the knowledge base are very critical operations. The above tasks become the responsibility of the knowledge engineer (K.E.). He will normally collect the knowledge about a particular domain by interviewing one or more humans considered to be experts in their field, although other useful sources of knowledge may be books, journals, etc. The domain knowledge will include facts, beliefs and heuristics together with rules of inference which will guide the use of the knowledge. The need for heuristics is very important since the problems faced by an expert often do not have an easily formalised or algorithmic solution. The ideas which underlie an approach to expert problem solving are shown in Table 1.

```
KNOWLEDGE = FACTS + BELIEFS + HEURISTICS

EXPERT SYSTEM = (KNOWLEDGE + INFERENCE RULES). INFERENCE ENGINE
    SOLUTION

                                        + EXPLANATION
```

Table 1

Having collected the knowledge, the next task of the K.E. is to decide upon a suitable knowledge representation structure for the knowledge base. Once complete it is important and necessary to perform some testing, since a number of difficulties may arise in acquiring the knowledge for an Expert System . The main difficulties are the human ability to express knowledge, and the difference between the way the human expert states knowledge and the way it is represented in the knowledge base. Other difficulties include the limits of Expert Systems technology, the complexity of testing and refining Expert Systems, and the disagreements between experts (3).

As a rule of thumb the collection of 75% of the knowledge will generally be completed in about 25% of the time it takes to collect all of the knowledge, but it should be borne in mind that the collection process is very time consuming and tedious, especially for large problem domains. Methods and techniques for automating the collection process would prove very useful, and considerable research is being pursued in this direction. Although some techniques and tools have been developed, it must be said that any significant progress will be a major advance in A.I. The need for good knowledge engineering cannot be stressed too strongly. Unfortunately the number of good, experienced, knowledge engineers is very limited, although growing all the time. This state of affairs does place a major constraint on the exploitation of Expert Systems.

SUITABILITY OF A TASK FOR USE IN AN EXPERT SYSTEM

A number of principles may be considered when deciding the suitability of a task for use with an Expert System. In many cases the principles are very similar to those which would be used by a designer when building traditional type software. The important principles are outlined below:

Select a Task which a Human Expert would not consider to be Trivial nor too Difficult

The problem is how do we define trivial or too difficult? It has been suggested that, using current techniques, trivial may be solved within minutes and too difficult would take months. This may mean that it would not be realistic to look at a complete domain, but rather look at a specific group of tasks within a particular domain.

Obtain the Help of a Human Expert who is Committed

Unless the knowledge engineer has a committed expert on whom he can rely and use regularly there is very little chance of success. Without an expert there is unlikely to be any way of obtaining sufficient knowledge required for a task. Without a strong commitment from an expert it will become very difficult to obtain the depth of knowledge and heuristics required to build an acceptable Expert System.

The Task Must be Defined Clearly

It is necessary to be able to specify the inputs and outputs with some precision. The expert should be in a position to outline the important concepts and relations, and the knowledge engineer should have access to a number of examples of problems and solutions.

Beware of Tasks Which Involve a Lot of Commonsense Knowledge

Commonsense reasoning is not easy to handle in present day systems. Attempting to build an Expert System which has expertise of several domains and is expected to use commonsense knowledge to go between them is difficult, mainly due to different paradigms and formalisms being involved.

In addition to the above principles it is also recommended that the following items of advice should be considered.

Have a Familiarity with a Number of Expert Systems

This will enable the knowledge engineer to be in a position to try and match the task to an appropriate architecture. Using an inappropriate architecture may make it more difficult to actually build the system, and may lead to efficiency problems at a later stage.

Final Acceptance May be Very Dependent Upon the User Interface

Any interface which is difficult to use or does not give explanations in a user desired format (structure and wording) is likely to make the use of the Expert System undesirable.

USES OF EXPERT SYSTEMS

Early Expert Systems were confined to academic research departments, and it is only since 1980 that Expert Systems have been used commercially. The first Expert System was DENDRAL (4) which works in the field of chemistry and is still used extensively. DENDRAL applies the rules of spectroscopy to spectrographs in an attempt to identify organic compounds. The rules are applied in much the same way that a human would apply them. Probably the best known Expert System is MYCIN (400 rules) which was developed at Stanford University by Shortliffe (15). It works in the field of medicine and in particular the area of diagnosis and treatment of meningitis. Several other early systems have been developed in the domains of chemistry and medicine.

Since 1980 the application areas have broadened. Two currently very important Expert Systems are PROSPECTOR and XCON (800 rules approx.). PROSPECTOR works in the domain of geology and is considered to be a specialist geologist in the identification and evaluation of mineral sites. Two important features of PROSPECTOR are the KAS knowledge acquisition system and its use of probabilistic reasoning under uncertainty.

DEC, the computer manufacturer, now use an Expert System called XCON. Their customer order department use XCON for checking VAX-II configurations. There are so many options available for configuring VAX-II systems that almost every one is unique. Prior to XCON being implemented it took approximately 2 man-days to check each configuration. The type of checking which has to be performed includes:

- checking power supplies are adequate;
- ensuring equipment will fit into racks;
- working out cable runs etc.

The same work is now carried out by XCON in about 7 minutes and DEC have admitted they are now dependent upon XCON.

A number of generic categories of knowledge engineering applications have been identified by Hayes-Roth, Waterman and Lenat (6) which are shown in Table 2.

Category	*Problem Addressed*
Interpretation	Inferring situation descriptions from sensor data
Prediction	Inferring likely consequences of given situations
Diagnosis	Inferring system malfunctions from observations
Design	Configuring objects under constraints
Planning	Designing Actions
Monitoring	Comparing observations to plan vulnerabilities
Debugging	Prescribing remedies to malfunctions
Repair	Exercising a plan to administer a prescribed remedy
Instruction	Diagnosing, debugging and repairing student behaviour
Control	Interpreting, predicting, repairing, and monitoring system behaviour

Table 2

WHAT DOES THE FUTURE HOLD AND DO WE NEED EXPERT SYSTEMS?

Expert Systems offer potentially significant increases in the development of user friendly computer systems. They are capable of explaining their actions to naive computer users, have the ability to learn from themselves or from teachers and generally to act more 'intelligently' than current systems.

With the current level of research being attached to 5th generation computers, which is all geared towards computers working as Expert Systems from the hardware, it becomes even more important to give Expert Systems some careful consideration. The plans of the Japanese 5th Generation Project indicate that the capabilities of these systems will be very impressive, particularly in understanding natural language and speech, and for being capable of performing new tasks either from plans generated by themselves or by being taught. The predictions are that these computers will be cheap and therefore available in large numbers. There is currently some debate taking place as to whether the Japanese will accomplish all of their aims set out at the beginning of the project. However, they have made sufficient progress to change the style of computing in the 90's from what we know it to be today.

Feigenbaum and McCormick in their book 'The Fifth Generation' (7) make a point which is worth noting: 'We do not know whether, even given the same heuristics that humans use, a system that can think faster and deeper will necessarily think down the same avenues that humans do. If it should go elsewhere, we do not know what lies at the end of such different avenues.'

What is also unknown is whether a machine can discover new knowledge, although the indications in this area are very favourable, particularly work being being done by Doug Lenat at Stanford (8). Again where this will lead is uncertain.

A number of important points make an Expert System potentially useful, including:

- Unlike a human who is mortal, an Expert System never dies, retires or leaves the company;
- The knowledge base may be expanded infinitely (theoretically, e.g. XCON now has 2,500 rules);
- Knowledge can be changed and updated regularly;
- An Expert System should be capable of explaining how it obtained a result - a human cannot always do this.

Consequently if you have any long term development plans for systems which are likely to be running in the late 80's and 90's, then you should be investigating Expert Systems.

References and Suggested Further Reading

1	BRAMER, M.A., 1983	'A Survey and Critical Review of Expert Systems Research', in Introductory Readings in Expert Systems. D. MICHIE (Ed). Gordon & Breach.
2	NILSSON, N.J., 1982	Principles of Artificial Intelligence. Springer-Verlag.

3	BUCHANAN, B.G. et al 1983	'Constructing an Expert System', in Building Expert Systems, Hayes-Roth et al (Eds). Addison-Wesley
4	BUCHANAN, B.G. & FEIGENBAUM, E.A., 1978	DENDRAL and META-DENDRAL: Their applications dimension. Artificial Intelligence 11, pp 5-24
5	SHORTLIFFE, E.H., 1976	Computer-based medical consultation: MYCIN. New York: American Elsevier
6	HAYES-ROTH, F. et al, 1983	'An Overview of Expert Systems', in Building Expert Systems, Hayes-Roth, et al (Eds). Adison-Wesley
7	FEIGENBAUM, E.A. & McCORMICK, P., 1984	The Fifth Generation. Michael Joseph, London
8	LENAT, D.B., 1982	The Nature of Heuristics. Artificial Intelligence 19, pp 189-249
9	NEWELL, A., 1973	'Production Systems: Models of Control Structures', Visual Information Processing, Chase W.G. (Ed) Academic Press, New York.
10	MINSKY, M., 1975	A Framework for Representative Knowledge. The Psychology of Computer Vision
11	GOLDSTEIN, I.P., and ROBERT, B., 1979	Using Frames in Scheduling, Artificial Intelligence: An MIT perspective 1, Winston & Brown (Eds), MIT Press
12	McCALLA, G. and CERCONE, N., 1983	Knowledge Representation. Computer, IEEE, October 1983.
13	SHORTLIFFE, E.H. and BUCHANAN, B.G., 1975	A Model of Inexact Reasoning in Medicine. Mathematical Bioscience 23, pp 351-379.
14	ZADEH, L.A., 1979	A Theory of Approximate Reasoning. In Machine Intelligence, Vol. 9, J.E. Haynes et al. (Eds) John Wiley & Sons.

M.J. Winfield
City of Birmingham Polytechnic
Computer Centre

A LIFESTYLE UNDER VM/CMS VSAPL

The talk was given by four staff from a major VSAPL site, which currently runs VSAPL Release 3 under VM/CMS on an IBM 4341 group 2 machine (8 megabytes, 1.2 MIPS), with around 30 APL users and normally between 10 and 20 concurrent users.

The item was introduced by Martin Malin, who expressed the aim of the talk as being to demonstrate that whilst VSAPL was certainly lacking the enhancements that many other APL interpreters made available, it was still possible to develop effective and attractive tools. The strategy required was threefold, namely to make best use of what facilities were available, to make up for the missing facilities as far as possible, and to exploit those features which were unique to the environment. The following three speakers broadly covered these areas.

First David Doherty talked about making the most of the screen-support facilities provided, namely AP124. He described in outline a parameter-driven package called SCREENIO (described in the article 'SCREENIO - An IBM Full-Screen Manager' in VECTOR Volume 1, No. 1) which had been developed to cover all aspects of full screen usage. The basic software provided by IBM gives utilities to assist with screen design, writing to the screen and reading from the screen. What was needed was an integrated facility covering not only those aspects, but also validation and assignment of input, and identification and actioning of input key pressed. SCREENIO permits design of screen layout, definition of both fixed and dynamic text to be written to fields, definition of validation and assignment rules for input fields, and PF key definitions and actions. In the application a single call to SCREENIO retrieves all the parameters from file, runs the screen and handles user entry as specified. Amongst the advantages of such a tool are easier application maintenance through much reduced application code and use of a standard, well-proven utility, increased speed of development, and consistent appearance and behaviour to the user.

Next Mark Longstaff talked about making up for missing facilities, such as a native APL file system. He described the standard cover functions developed for AP110 to simulate a component filing system as in other interpreters, as well as giving access to fixed or variable length CMS files. Again the advantages stem from standard utilities giving speed of development and ease of maintenance. Mark then described two facilities which permit non-terminal execution of APL routines. The first allows detached running of non-interactive tasks, such as generation of lengthy reports, with status reporting to any other logon ID as execution proceeds. This frees the terminal (and the user!), but still consumes CPU at peak time. The second allows overnight execution of non-interactive tasks by punching jobs to the virtual reader of a machine which is scheduled to log on automatically each night and run through the night if required. Thus although VSAPL does not directly offer batch or non-terminal tasks, it is possible to accomplish both using AP100 and AP101.

Finally Phil Last described a full-screen workspace manager which makes extensive use of the alternate input stack (AP100) and the full-screen editor (XEDIT). It provides a wide range of facilities for selecting, editing, grouping and listing functions and/or variables. There are also string search and/or replacement options which use XEDIT and run many times faster than any comparable APL algorithm, as well as execution of system commands and various workspace housekeeping utilities. Thus use of facilities peculiar to the environment have allowed the development of an extremely powerful and comprehensive tool.

In closing Martin summarised by saying that, despite the well-known limitations of VSAPL , thoughtful and intelligent use of the available facilities had enabled a reasonable simulation or equivalent of many of the enhancements available elsewhere, and some that weren't.

The questions which followed elicited details of the hardware configuration, which is as above, and discussed some of the more obscure peculiarities of AP124 and XEDIT.

Richard Nabavi minding his QLs.

Somes APL people (two out of three anyway). Tim Perry and David Alis on station.
1 0 1\APL[2 1]

What's New for 1985?

Organiser's Report by Dick Bowman

Awakened from my reverie on Thursday 6 December by the delicate plop of mail in the intray and pleased to find therein an invitation by someone who'd prefer to remain nameless (they'd offered to teach me APL a few weeks earlier) to come along to the "What's New for 1985" show on Monday 10th. I'd rather thought I might go anyway, but it's always nice to find someone cares.

What we'd set out to do with this show was part of the Association's quite deliberate policy of moving APL away from the introspective ghetto of talking only to ourselves and creating an image which meant that only APL converts would want to be there. So goodbye to the basement room of Imperial College (home of past incarnations) and to the shanty town of corrugated cardboard stands which we'd become accustomed to. The rather plusher surroundings of the Regent Crest Hotel beckoned, we bit the bullet of substantially increased costs of running the show (or, more accurately we bit our lips while our bold exhibitors decided to pay the substantially increased costs).

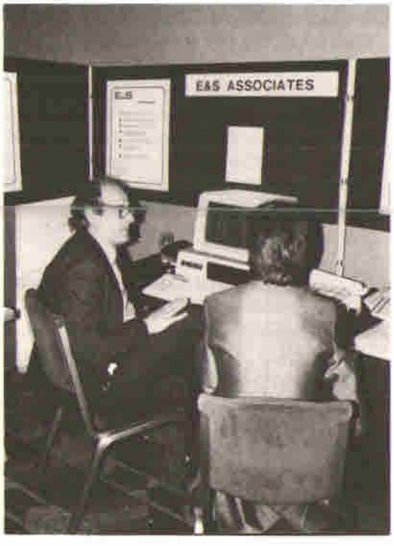
At the end of the day we had a very fine-looking exhibition (and if I single out I.P.Sharp Associates and Dyadic Systems as particular contributors to the visual excellence it is only in recognition of the degree of extra work which their 'permanent stands' represent); we also had a programme for the day in which Product Forums ran throughout and a fair smattering of new products (the accompanying photographs and reports will tell you more).

Attendance was very respectable at around the 100 mark (plus stand staff, representing the diehard core of the Association); we signed up 17 new members and had the opportunity of reminding a few more about all the money they still owed us. As always, we'd have liked to see more and can reflect on the qualitative difference between 100 people all sat down in one room at once and the same number of people strolling in and out throughout the day. Personally (with a degree of bias) I felt that there was ample opportunity actually to talk to the exhibitors and that the whole day was one in which everything was relevant - unlike certain other cattle markets of the computing exhibition industry.

In the longer term, we intend to continue providing the mixture of the more usual technical meetings and this sort of more overt commercial occasion. With APL86 coming closer we can look back on this event and the Loughborough exhibition as giving grounding in the important matter of putting together an attractive and professional show. No doubt there will be some commercial event organised by the BAA between now and 1986; quite what format this will take we haven't yet decided.

Finally, I feel that it is important to recognise that putting on events like this is more than a one-person effort and in addition to the efforts of the exhibitors I would like you all to recognise the sterling services of David Allen and David Preedy (Product Forum chairmen and vendor chasing), Roy Tallis and Panos Anagnostopolus (for their ability to extract money from ex-nonmembers), Phil Goacher and Neil Truby of the BCS for their advice, experience and guidance. If I appear to have omitted Stan Wilkinson and Dominic Murphy of the Activities subgroup this is only because we jointly felt that their non-involvement would be a clear indicator of there being no possibility of any conflict of interests (Stan's degree of participation in kicking off the Product Forums overshadows all of us, of course).

As the afternoon wears on, delegates take their chance to buttonhole the exhibitors, to visit the bookstall, and to enjoy a quiet cup of coffee in the foyer.



Selected Highlights

by Adrian Smith

Gareth Brentnall (APL*PLUS)
Compiler Support for APL

Oddly enough the compiler is written in APL! There have been numerous attempts in the past to allow some kind of partial APL compilation. This is the latest, and sounds as if it may be close to success. The effect of compiling functions is almost invisible to the user; essentially they behave just like locked code except for a possible speed-up of between 3 and 10 times. However there are (as there had to be) a good many snags, and I cannot see it ever being a cut-and-dried decision to compile code routinely.

Firstly if you want maximum benefit from compilation you really need to DECLARE (shades of my PL/1 days) all your local variables. Secondly compilation has no effect on transactions with files or other auxiliary processors. In my experience this is where the majority of looping occurs in APL, and would have been an area where compilation could have paid very big dividends.

Thirdly the actual compilation costs you an arm and a leg, so you really do need to be hammering the CPU before it pays. Of course you also lose the instant error diagnosis, as the line numbers no longer exist. Clearly you would need to keep a parallel 'debugging' copy of everything just in case of disaster.

However having said all that, there are clearly people who need to solve FORTRAN-type problems (Linear Programming, Simulation etc.) in APL, and anyone who is doing this kind of thing may find that partial compilation is just the service they need.

Richard Nabavi (MicroAPL)
APL on the Sinclair QL

As far as I know this was the only talk to get a spontaneous round of applause at the end! Richard began by musing on the unpleasant question of why, if APL is so wonderful, is no-one using it?! A slight exaggeration I know, but not too far from the truth once you start looking at schools and the home computer. He felt that the three outstanding reasons were:

- Price. The hardware is all well above the top end of the home market, and the APL itself is not cheap in these terms.
- Off-putting appearance. Richard quoted one QL user ...

  "Since starting to use QL/APL, I have become convinced that the conventional character set was just an elitist trick to restrict the spread of an immensely powerful language"
- Non-standard peripherals. There are all sorts of horrid snags as soon as you hook up APL to normal off the shelf devices. For example negatives turn up as '@' on HP plotters. Cost is another penalty of being out of the mainstream; an APL VDU will set you back some £750, compared to £350 for a standard version of the same device.

APL/QL is an attempt to take these issues full in the face; in other words to produce a cheap APL, running on a normal keyboard, with no peculiar hardware requirements. How well MicroAPL have succeeded you must judge for yourselves; in general I found the choice of keywords quite effective, but was rather distressed by the refusal to allow ' = ' for assign (if you're going BASIC you might as well be hung for a sheep as for a lamb!) and by the ambiguous use of '-'. I suspect that the first time user who gets a SYNTAX ERROR with '3-2' might have cause to wonder about the supposed consistency of APL.

Fast and cheap it certainly is. In a very brief encounter with a QL I found it very speedy indeed; nearly half as fast as top-line systems like the SAGE, and a lot quicker than the IBM PC. Even the microdrives may prove less of a disaster for APL than for other languages, after all we tend to)LOAD everything into memory and rarely need to venture outside the confines of our nice comfy RAM until lunch, when a quick)SAVE is all there is to it.

At £99 (VAT included) you can't go far wrong (assuming you already have the QL!). Even with £200 on top to get a reasonable workspace, this really does look like a mass-market product for the first time in the APL world. No wonder it drew such an enthusiastic response from the biggest audience of the day.

John Ward (APL*PLUS)
APL*PLUS for UNIX

Again this attempts a partial compilation, this time automatically on the first pass through each function. John claimed a speed-up of around x2.3 on subsequent executions. It allows access to standard Unix drivers from within APL (for jobs like setting up the keyboard), and maps the APL system commands, such as)LIB, to the Unix directory structure.

Neil Macmillan (APL*PLUS)
APL*PLUS Release 4.0

This is covered in detail in 'News from Sustaining Members' so there would be little point in my saying more here. The talk was received with considerable interest, and the idea of using APL*PLUS on the IBM PC/AT obviously excited many of the audience.

The Information Centre and Changing Technologies (The IP Sharp APL Users' Meeting) 17th - 19th October 1984, Toronto, Ontario

Notes compiled by Adrian Smith

Introduction

By a stroke of good fortune Rowntrees had already asked me to spend some time at our Canadian factory in October, so I was delighted to combine business with business and book a place at my first IPSA international conference. I was even so bold as to offer my services for Ken Iverson's panel on 'Mice vs Menus vs Commands', and was really delighted when he took me up. In all this was a most interesting and enjoyable event, and it was attended by over 320 people from 17 countries. Most of the papers are available in printed form in the conference proceedings, but the contents of both Ken's panel and a similar discussion on Information Centres will not be published by IPSA. Consequently I will simply give my impressions of the main talks, and will include full details only of such material as would otherwise languish forever in obscurity.

People of the Global Village

Jim Cunnie (ITT)

This was the first of three 'keynote' sessions which began the conference. I found it a very frustrating 45 minutes, as the speaker whipped through sheaves of unreadable foils, flinging out unsupported propositions at a rate no-one in the audience could possibly absorb. His basic message seemed to me to be "It'll be alright on the night!" Never mind the problems of today - population growth is an intrinsically self-regulating system, we are already past the peak growth, and by the mid 21-hundreds the mean GNP per head will settle out at £5,000 - £10,000, with a worldwide population of some 15 billion.

The basic culture will be scientific (with the emphasis increasingly on the 'analytical') and will be heavily dependent on telecommunications. The visible processes of government will matter less and less as we move into an age where global 'face-to-face' interaction becomes the norm.

Hence the phrase 'Global Village'. As with all such 'take it or leave it' presentations it was impossible to decide whether there really was background research behind the glib facade. I suspect there was, which made it all the more of a pity that no attempt was made to put it across.

Introducing the Global Information Centre

Lib Gibson (IPSA)

The speaker stressed the need for firms to deal with planning problems of worldwide scope on shorter and shorter timescales. In order to ensure the integrity and timeliness of data across the world there is an increasing demand for easy *transparent* telecommunications, and for software which will make amateurs self-sufficient. This must work with data from any source, and must obviously be both responsive and easy to use.

The rest of the talk was really a (wholly excusable) sales pitch, so readers are best directed to the IPSA handouts at this point. I think the answer to your prayers is called 'Viewpoint', but you must judge for yourselves!

A PRACTICAL INTRODUCTION TO EXPERT SYSTEMS

by Christopher Harvey

OUTLINE OF THE PROJECT

The Problem

The computer press is full of stories of "Intelligent Knowledge-Based Systems" and "Expert Systems". Industry seems very excited about them but just what are they?

As part of an Artificial Intelligence section of my Degree I had been introduced to Expert Systems by Mike Winfield, a lecturer who subsequently became my supervisor for the PATTIE (Pipe Analysis Techniques Through Inferential Expertise) project. In common with many people, the more I read, heard and thought about what Expert Systems were and what they could do, the more their potential for use struck me. I had to find out more.

The Proposed Solution

It was decided that an excellent way to solve the above problem was to actually develop an Expert System. In the same position as myself (excited but ignorant) was a lecturer in the Mechanical Engineering Department called Dr. Steve Douglas.

The obvious struck us: for my final-year major project I could develop an Expert System for, and in conjunction with, Steve Douglas. The system could be in a mechanical engineering vein and would serve the purpose of an introduction to the field of Expert Systems for the both of us.

The initial aims of the project were stated to be: to act as a basis for understanding the following areas: Knowledge Engineering, Knowledge Representation, Natural Language Interfaces and Inferencing Mechanisms.

HOW PATTIE CAME ABOUT

The Choice of Domain

The next problem was to find a domain around which to design and implement a system; that is to say, for which aspect of Mechanical Engineering could we write an Expert System? (We were lucky to be in the position of being able to choose the subject domain around which we could write our system. This would not, of course, normally be the case!).

After some thought and after considering areas such as Bearing Analysis and Mechanism Theory we came up with Pipe Analysis.

When first-year undergraduates of Mechanical Engineering start their courses there is an initial 'barrier' to be got over as regards just what is involved when considering piping from the points of view of design of pipes, what types of material to be used, what types of fluids, gases (the "transport mediums") can actually pass through those pipes, how those pipes have to be jointed, how they can be fixed to their surroundings, etc.

Therefore a system to aid first-year undergraduates to overcome this barrier would be of some use. This seemed a reasonable idea on which to build.

... and the spreadsheet will respond. The possibilities are anything but limited, and users brought up on this style of system are unlikely to be very pleased when offered the old-fashioned reactive approach on the mainframe.

Bob's next distinction was between 'segmented' and 'apertural' systems. I think that the basic difference is that if the designer fixes the hierarchy of functionality (e.g. by a menu-tree), often embedding artificial distinctions, say between 'Update' and 'Display', then the system is definitely segmented. By contrast the apertural system makes all commands available all the time, and lets the user say how they are structured. He can likewise scroll at will around his data, slicing and re-ordering as it suits him. Once again the spreadsheet is the archetype.

Bob gave some very clear design principles which we should look for in such systems, and then moved on to discuss the possibilities of natural language processing. He was very much of the view that 'true' natural language is rather a pointless quest until we have managed to get voice-recognition to a workable state. Even then it is unlikely to be of much use in the most touted application of today - database query. To get precise answers we unfortunately need to ask precise questions! In conclusion, the 'English-like' approach of most of today's 4GLs is probably quite satisfactory when you are searching for enhanced user-productivity on today's computers.

In the second half of his talk Bob covered much the same ground from the point of view of 'how would we do it in APL?' The section on integrated data was somewhat Sharp-specific, and rather beyond my comprehension. Interested readers should dig up the original, where full function listings are given. Of much more general relevance was the detail on proactive systems; specifically on ways and means of building command languages in APL. Three approaches were given, in increasing order of sophistication :

- Keyword matching. This is the most basic type of language, suitable for jobs like scrolling around your data. You can say things like 'UP', 'DOWN 10', 'LEFT 3 PAGES','TOP'. Normally such a language would supply sensible defaults, and tolerate unique truncations.

- APL syntax analysis. The logic of this technique is that the APL interpreter is itself a very effective syntax analyser, so it seems daft not to use it. If the commands can be made to look like valid APL expressions such as 'SHOW DATA WHERE (AGE > 35) AND (SALARY BETWEEN 12000,15000)' then simply use APL to execute them. Any SYNTAX/VALUE ERRORS can easily be trapped and reported. The functions would probably in fact be dummies, building up an easily executed command string for a master driver to activate.

- Augmented Transition Networks. Enough said. If you want to get that deeply into formal grammars then these are probably what you need - the IPSA proceedings include a lot of helpful functions to get you going.

Finally Bob mentioned that AP124 (along with some helpful utilities) can be effectively used to tackle the apertural approach. This contention had the look of a one-paragraph afterthought, as did the ultimate conclusion that Sharp APL was a 'very congenial environment' for implementing apertural/proactive systems. This very minor quibble apart I could hardly agree more with the view that users really do get more productive if they are given integrated data and non-procedural processing. To take best advantage of modern computing we need to move away from the reactive/teletype/segmented approach towards the two qualities Bob dubbed 'proactive' and 'apertural'.

Mice, Menus and Masters in Application Design
Panel Discussion chaired by Ken Iverson

Jo Sachs (Morgan Stanley) Gosta Olavi (S E Banken)
Adrian Smith (Rowntree Mackintosh)
Professor Donald McIntyre (Pomona College)

The format of the meeting was a series of (more or less) formal talks, followed by questions from the floor. Each of us spoke for about 20 minutes, and there was a marked reluctance to get too deeply involved in any sort of real debate.

Jo Sachs came down on the side of commands, but with the proviso that 'use once' systems were obviously better driven by menus. He was perturbed by the tendency of menus to influence the user's perception of his problems, and also by the lack of ready extensibility in most menu-driven systems.

Gosta Olavi very thoughtfully provided a complete text of his talk, so I shall simply refer readers to our general papers, where it is printed in full. He felt that (contrary to popular opinion) command languages do not in fact stimulate experimentation - users stick rigidly to what they know! He was more in favour of well designed menus where all the options were attractively offered.

Next in the list was your esteemed editor, who (very sneakily I thought) sidestepped the whole issue by declaring that the exact style of dialogue was quite irrelevant - the important thing was its ability to learn from the user. In a total abuse of editorial privilege I am also going to include my own notes under the general papers heading, so I now pass quickly on to the final talk from Professor McIntyre.

He took a very different tack from the rest of us, and started from the basic premise that the spreadsheets (and Lotus-123 in particular) must be doing something right! He drew on some quite detailed experience of Lotus in giving us examples of array handling and table generation. It was fascinating to see the kind of obscure and arcane things people get up to - Lotus macros make APL look almost readable!! It was also slightly disturbing to discover that I found some of the more convoluted bits of spreadsheet manipulation easier to follow than the 'simple' APL equivalent. Perhaps I am still a BASIC programmer at heart - awful thought!!

Mice got very little coverage at all in the debate, except for a general view that they simply put a seductive gloss on menu systems. Since menus came out of the session second best, the poor mice had little chance. No-one felt that 'natural language' would have any great impact in the near future; in particular Jo Sachs pointed out that it would be of little value without voice input.

I think that that just about wraps it up. Needless to say it is much harder to take proper notes when you are one of the participants, and I am sorry I have been unable to do full justice to a very lively and interesting discussion.

A James Martin Lecture
James Martin

I know this talk had some fancy title (Global Information Centres figured in it I expect) but it was really just another typically Martin tour de force. As an object lesson in holding an audience for over 3 hours it was brilliant; whether it was more than just clever words well assembled you must judge from my notes. I shall try to reproduce as much as I can, but there will inevitably be quite a few inaccuracies in the diagrams - you can only sketch so fast.

The first half hour was devoted to a lightning tour of the latest advances in hardware. Computer-aided design has made it possible to pack more and more components on a chip, and is thus helping to reduce the number of external connections (the expensive bit). Gallium arsenide will speed up processing by a factor of 10, allowing the optic fibre to come into its own at bandwidths in excess of 10 GigaHertz. In fact much of the laboratory stuff looks most at home in the PC, and some form of Personal Satellite Earthstation will soon become a real possibility. The big unsolved problem is in the use of parallel computing; 10 M68000s cost vastly less than the same amount of 'big machine' power. If only we knew how to get them to co-operate!

A couple of interesting asides: 'When men last walked on the moon, we had yet to invent the micro'; 'There are no micros on Concorde - in fact there would be room for another 10 to 15 passengers in the space taken up by her racks of obsolete computers!'

To sum up the first section; there will be advances in all three key areas of computing technology:

- processing. Reduced instruction sets, and hard coded interpreters (PROLOG-on-a-chip) will speed things up even without co-operative computing.
- communications. Light is currently giving us around 10^9 bits per second. 10^{13} is certainly possible quite soon.
- storage. You can get 2 Billion bits on a standard Compact Disk (the shiny sort used for classical music). These cost 25p each to make. Enough said.

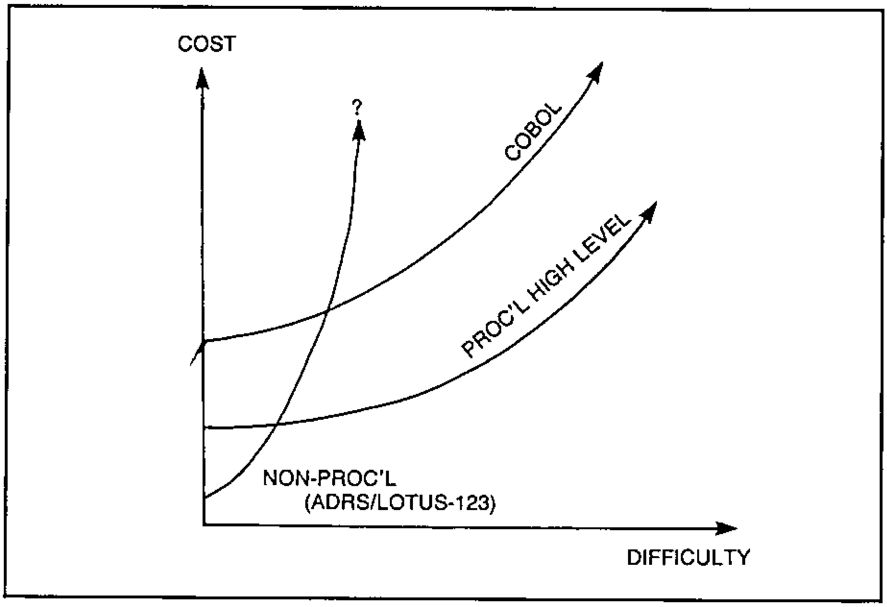
Next we come down to Earth with ..... Data Processing. What chance has the revolution when 80% of DP shops are still writing spaghetti COBOL? Why do they still refuse to accept the proven savings of 4th Generation Languages?? In fact why are they so downright irresponsible with their company's money!? (The speaker got quite warmed up at this point.)

End users are the best hope; it is they who are forcing the pace by demanding computing facilities which get them results with:

- minimum work.
- minimum skill.
- no alien syntax or mnemonics.
- fast prototyping regardless of machine performance.
- 'up front' error checking.
- minimum maintenance.

As more and more packages hit this market, it is slowly becoming clear where their relative strengths lie. As systems get harder the cost curves behave somewhat like Figure 1:

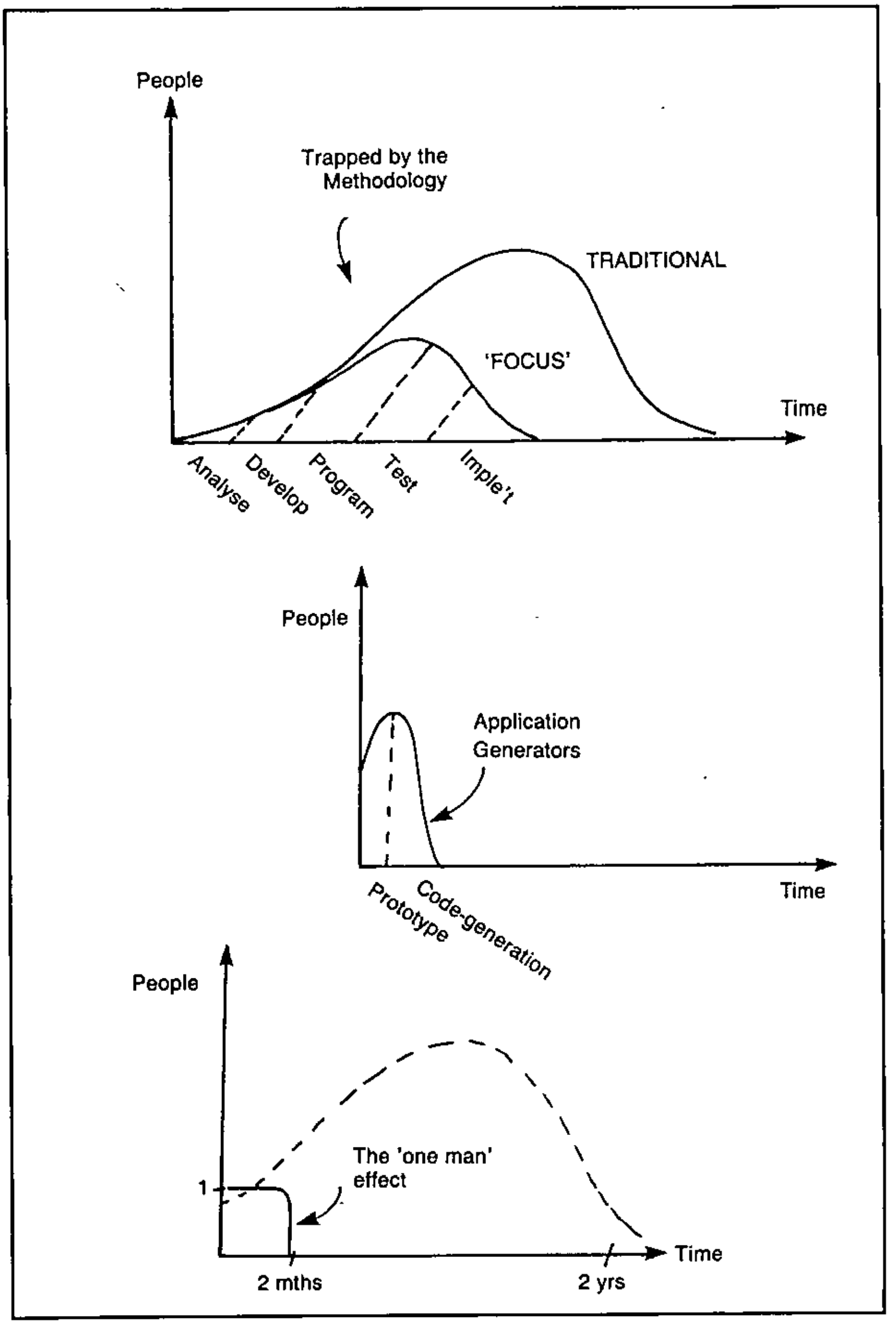
Fig. 1



... but we are really only scratching the surface of true automatic programming, and the use of CAD techniques by systems analysts will begin to eliminate the whole tedious cycle of specification and programming.

In fact the impact of 'Fourth Generation Languages' (FOCUS and the like) has been hardly noticeable - that DP backlog is still there, and is likely to remain for as long as managers stay with today's methodologies. True, the people-cost does come down, but nowhere near as much as it could (Figure 2):

Fig.2


People
Trapped by the Methodology
TRADITIONAL
'FOCUS'
Time
Analyse
Develop
Program
Test
Imple't
People
Application Generators
Time
Prototype
Code-generation
People
1
The 'one man' effect
Time
2 mths
2 yrs

The most dramatic gain occurs when the breakthrough to a 'one-man' project is made, but even for really big jobs the benefits are there for the taking. On a rough scale where one 'function point' is about 100 lines of COBOL, we still find that the picture looks like Figure 3:

Fig.3

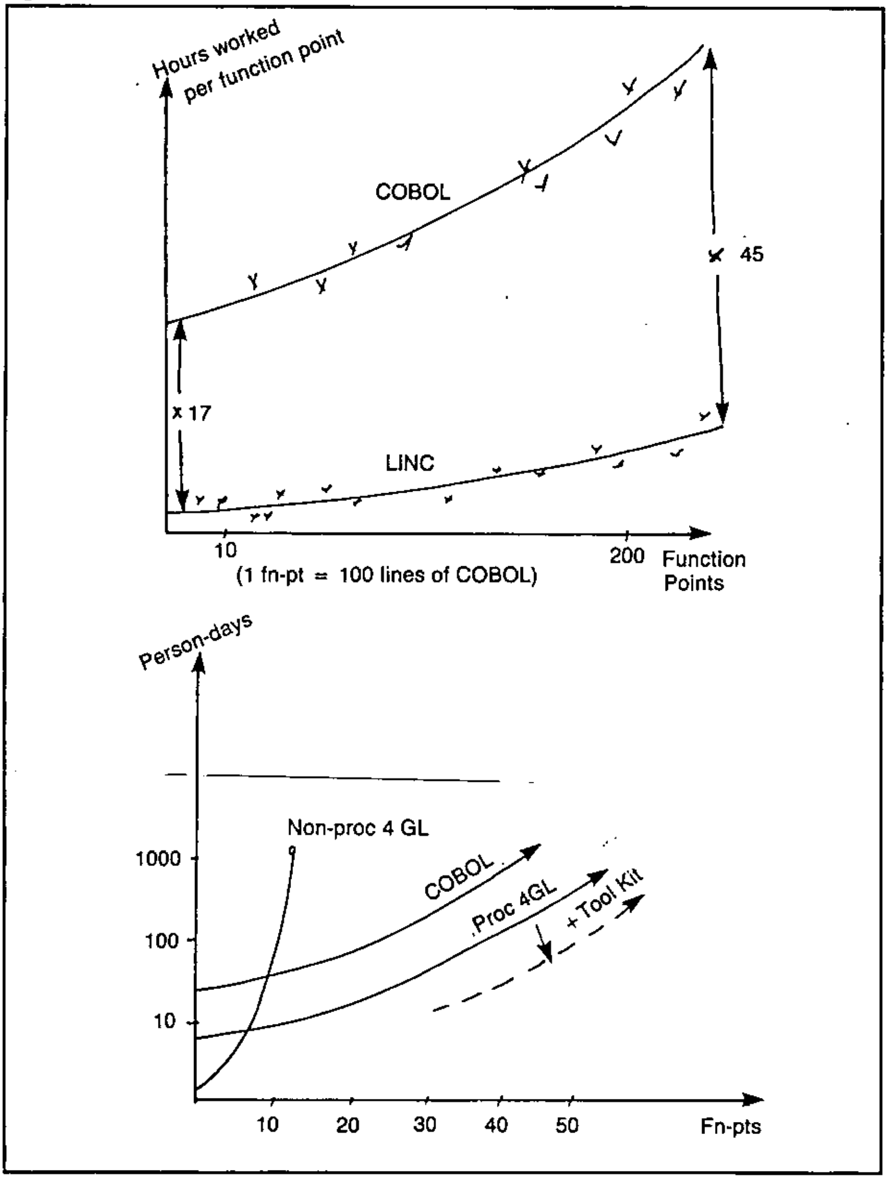


Fig. 4

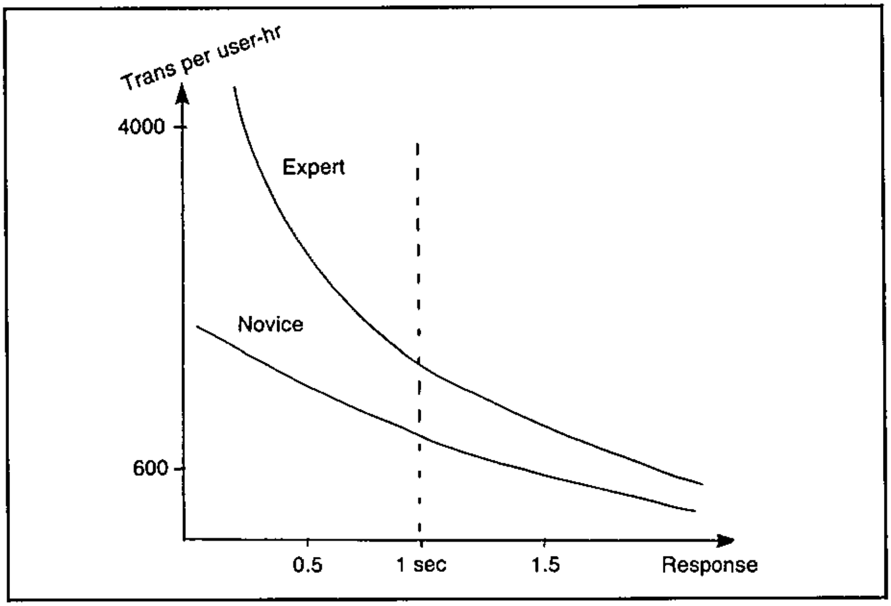
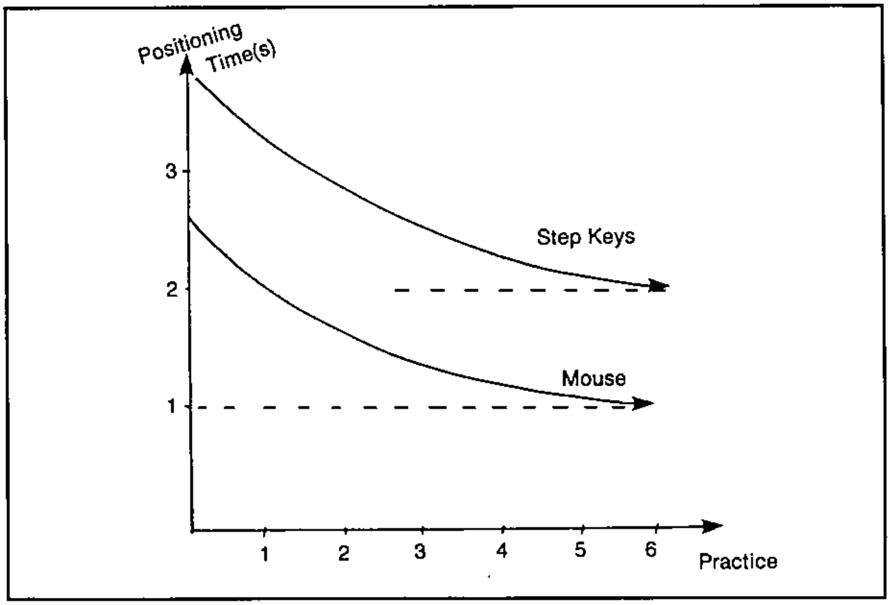

Trans per user-hr
4000
600
Expert
Novice
0.5
1 sec
1.5
Response

Fig. 5


Positioning Time(s)
3
2
1
Step Keys
Mouse
1
2
3
4
5
6
Practice

So much for the DP shop - what about the users? We now had a short interlude on response times, mice and 'human factors' in general.

The following two graphs say it all. Faster systems really do matter, and the decay time of the human short-term memory (roughly 2 seconds) is the critical cut-off (Figure 4):

In a typical case, average response time was cut from 2.3 sec to 0.84 sec, which gave a 52% increase in productivity, a 113% improvement in 'function points per error', and a cut in the cost of programs by 37%. Mice too can be worth the cost, simply by reducing the time it takes humans to position a cursor (Figure 5):

The problem is one of getting people started; to work a mouse really fast you do need to practise, and there is quite an initial barrier to be overcome. (Interestingly, this was almost the opposite view to the consensus in Ken's discussion group. Perhaps I could solicit some letters on this point?)

Onward to Information Centres and some thoughts on where these might be going. The Bank of America suggest the following as the most critical success factors:

- 'high profile' top management support.
- ease of use.
- worldwide availability and ready access.
- excellent training and support.

... but there are also some dark clouds on the otherwise rosy horizon:

- in general the DP backlog has not been reduced.
- user-demand was greatly underestimated; controls on growth and on extravagant use of computer resource have often been 'retro-fitted' much too late.

However in spite of these worries it looks as if the Information Centre boom is set to continue at an ever increasing pace. There are some 30M 'knowledge workers' in the USA, of whom 50% will probably have some form of workstation within the next 3 years. At a conservative ratio of 50:1 that still represents a demand for 300,000 consultants!

If you don't believe it, just look at these three alternatives, all of roughly equal cost:

- 50 workers.
- 49 workers, each with an IBM PC.
- 48 workers, each with an IBM PC, plus 1 IC consultant.

Faced with that choice, few companies would be daft enough to pick anything other than the third option.

Finally (and I really do mean that) a most invigorating afternoon came to a close with some pertinent comments on AI, and how we might one day make money from it.

Just what are the Japanese driving at with their '5th Generation'? Some idea can be gained from a comparison between people (who can manage a paltry 2 Logical Inferences per Second) and the Japanese target of 10^9 LIPS by 1990. Basically they are going for the mass-market in Expert Systems, which means systems where human expertise can be neatly packaged into consistent rules.

Typically such systems will be used to improve the performance of technicians (e.g. judges), to aid top experts (medical diagnosis), and to take over from people in situations where our speed of response is inadequate (real-time financial trading). They will then move out into the vast untapped home market, probably via those miniature optical disks, to advise us on such vital day-to-day activities as the care of our pot-plants. Sic transit gloria mundi.

Supporting Information Centre Users

Lael Kirk (IPSA) Chris Baxter (Midland Bank plc)
David Crossley (Canadian Imperial Bank of Commerce)
Edward Kohler (Xerox) Gladys Lim Eng Choo (Singapore Airlines)

And so to Day-3. This was by far the best attended session of the morning, and was again in a Panel Discussion format. Where possible contributions are preceeded by the speaker's initials, so I hope some consistent threads will emerge from the notes.

Question:	*How do the panellists go about marketing their respective Information Centres?*
DC:	Business is mostly by referrals - the IC staff have no real commitment to marketing.
GL:	Using an in-house magazine, and with regular seminars.
EK:	In the early stages (8 years ago) it was a matter of knocking on doors, but now a quarterly newsletter is sufficient.
CB:	Just getting under way, with 'house calls' to senior management, and some notes in the head office bulletin. The biggest problem is to back off until capacity can cope.
Question:	*What kind of newsletters do you all use?*
DC:	Most articles are by Information Centre staff - it is hard to get users to contribute.
GL:	Regular feature in an existing house magazine.
EK	Some intimidation yields enough articles for a quarterly journal. It helps to have a really professional layout and publishing service available.
Question:	*Where is the best place to site yourself?*
CB:	We are lucky enough to have a nice plush demo / walk-in area studded with micros, terminals, plotters and audio-visual equipment. The plushness matters if top management are to feel at home!
EK:	Much the same idea, but rather plainer. It is vital to pick a site in a 'high traffic' area, not tucked away in the depths of DP.
Question:	*Has anyone done any market surveys?*
EK:	We support around 5,000 active accounts. A 10% survey got nearly a 70% response, and the overall saving suggested was some $10M p.a.

DC: 1,200 people were invited (over 600 actually turned up) to an open-house. N.B. Make sure you get names and phone numbers from as many people as you can!!

Question: Do you do any sort of 'needs analysis' when approached?

CB: Top management are usually without any computer support at all. Many jobs turn out to be little more than simple collation and reporting. If a proposal starts to smell like an 'operational' system it should be avoided like the plague.

EK: 'What data should we extract' is usually the first question.

GL: 'Which package, and will we need much one-off APL?'

DC: 'Is it ICU, SAS, EASYTREIVE, ... etc'

Question: Wot about APL then?!

DC: 40% - 50% of their IC users are 'dependent', i.e. they draw the line at anything more demanding than pushing PFkeys. 40% - 50% are 'independent'; they are happy with ICU and the less procedural bits of 4GLs. Maybe 5% are 'expert'; they will get stuck in to SAS, APL, even PL/1.

GL: Nearly all are either independent or expert; certainly the majority write their own APL with little assistance.

CB: Moving towards local networks linked to the Information Centre. APL development is often contracted out to recommended consultants. This way the IC doesn't get lumbered with maintenance.

Question: How much work does the PC handle?

DC: Lotus-123 is very common, but not supported through the IC.

GL: VisiON, Execuvision, Symphony are all supported.

EK: Lots of Xerox PCs turning up, but no software support through the Information Centre.

CB: Out of 8 staff 4 are on the mainframe side, and 4 are dedicated to PC work with MultiPlan, Lotus-123 etc.

Question: How does the Information Centre charge (if at all)?

DC: They don't.

GL: Internal charge on CPU and file space.

EK: Nothing for labour, so again all costs are recovered on computer load.

CB: 'We charge for everything!' PC software is bought at discount and charged at retail!! £2M is a lot to get back, so every penny is welcome.

Question: *Can the Information Centre cope with the impact of users doing their own APL?*

EK: A (public) list of the top-10 most hated applications works wonders.

??: There was once a FOCUS program that ran for 134 hours; it isn't only APL you need to worry about.

GL: Users must return regular status reports. Also the monthly billing is monitored for any oddities.

DC: 'We just let them get on with it'. The biggest user is outside Information Centre control anyway.

CB: So far the system has proved largely self-regulating. Monthly billing has a high profile, but the 'quiet word' is used as a last resort.

Question: *How does the panel help users to cook up benefits cases? (I don't suppose for a moment that it was really worded like that, but my notes are less than explicit on this point.)*

DC: There are some quite elaborate cost/benefit procedures to follow, so the Information Centre staff can be very helpful with the protocols.

CB: We estimate the cost, and the user must then justify it. Generally they are not particularly professional about this, so it can often be hard to show concrete benefits for the Information Centre.

GL: No charge is made for any of the 'walk-in' facilities, but cost/benefit cases are required for any hardware.

Question: *Could the panel discuss any problems they have found in getting at corporate data.*

DC: Data is generally departmental - there is no comprehensive database. Other problems are the physical transfer of tapes to the IC computer, and red-tape in the systems area.

GL: Data can be pulled out from the IMS database, but this needs co-operation and careful organisation.

EK: Xerox has many different branches, each with its own database. Data can be shifted around using a fast hardware link, shared disks and purpose-built software. Currently they manage some 6,000 transfers each month.

CB: Split sites are again a problem. Data pools (some 8M records) are copied to the IC via overnight export.

Question: *What training should the Information Centre offer to users?*

CB: Mostly 'on the job' with some workshops. However there is an increasing need to teach basic keyboard skills, word-processing etc. Here the IC goes for 'off-the-shelf' courses and is looking hard at Computer-aided Instruction.

EK: Generally depend on the software vendors, and from then on users increasingly train each other. All trainees get 'free time' in which to practise.

DC: Classroom courses (up to 3 days) are given by a separate training department. Some self-study (SAS/DCF), but no in-house APL courses due to lack of terminals.

Question: *Where do the Information Centre staff come from?*

CB: Avoid DP and get good graduates directly. Training stays well away from DP concepts, but includes the IPSA APL course, manning the phones, and generally learning on the job.

EK: Mostly they are DP'ers who have seen the light. This helps as they generally know which strings to pull!

DC: It is much easier to train a banker about packages than a programmer about banking.

Question: *Does the Information Centre do any programming?*

CB: 'No' to tailor-made systems. 'Yes' they will do general-purpose software, but otherwise they use contract programmers only.

GL: Users get help with their first application only; from then on they are on their own, but with a good example to follow.

EK: 'It must be going on, but I don't look too hard'.

DC: The original notion was 'no programming', but quite a lot of SAS and some APL is now being done. In fact SAS is threatening to turn the whole Information Centre into a programming shop!

Question: *Do they run a 'hotline' or help desk?*

All: Yes, and it pays to log calls and be very conscientious about getting back to the callers.

And so to lunch.

Closing Remarks
Ian P Sharp

You may have noticed that APL has been given rather a low profile in the majority of papers. In fact several delegates were heard to complain in no uncertain terms that they had come for the APL, not for all this guff about users, data, 4GLs and the rest. Ian Sharp's closing address was in some ways a justification of the (non-APL) theme; yes of course IPSA remained committed to APL, but primarily as a means of making available their databases and the software to access them.

No-one makes money any more simply by selling APL; you must use APL as a means to an end, and the end is best achieved through packages whose APL content is well and truly hidden. That seems to me to have been the outstanding message which emerged (by no means unopposed) from a very stimulating 3 days.

GENERAL ARTICLES

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

COCKING & DRURY LTD.
THE APL PROFESSIONALS

Enhance your APL Information Centre

An enhanced APL will improve programmer productivity — but what if you can't afford to be cut off from future IBM Information Centre products?

Cocking & Drury have a range of powerful enhancements to VS APL that will not commit your company to a non-IBM strategy.

* Multi-user APL filing system
* Enhanced report formatting
* High speed printing
* ~~Mailbox facilities~~
* Execution of non-APL programs from APL

For further information call us on 01-493 6172 or write to 16 Berkeley Street, London, W1X 5AE

Preferring a Menu-Oriented Dialogue

by Gösta Olavi

WHAT IS A GOOD DIALOGUE?

What is a good user interface? I am currently leading a study at Skandinaviska Enskilda Banken, that aims at producing an "APL Dialogue Standard". The study raises questions also penetrated in this paper.

★ The ultimate goal of the study is of course to enhance user productivity.

★ The suggested dialogue scheme should be so widely applicable that the user meets a similar behaviour in all APL systems.

★ Many of the employees in my company have day-to-day experience with dialogues in the several transaction-driven applications. Most of these are available in an IMS/Cobol environment. One aim of the study is to see how APL dialogues can be made similar to IMS (and other) dialogues, at least not introducing differences where there is no real reason for them.

★ Another aim is to see how, within reasonable limits, line-by-line command-oriented dialogue and menu-driven, 3270 fullscreen applications can be given similar features.

★ How to design a certain dialogue depends very much on: user experience; nature of the task to be performed; and the user's equipment. These will be studied in the next section.

User experience

The novice appreciates good advice from the system, and benefits from a helpful, explanatory, menu-driven dialogue. The experienced user gets tired of all helpful texts, and wants less control by the system. He can use a rich command language, and wants a facility to re-use command sequences that proved useful in previous work sessions.

Any system will always have novice users - everybody is when first using it. Many users never accumulate experience, since in their eyes the system is a minor tool, which lets them perform their task with less paper work. In my company we design most systems to suit the many infrequent users.

The novice in one system often has experience of other systems, and appreciates standardisation of dialogue between applications. And, since very many users are also users of the fullscreen, menu-oriented IMS applications, we often design APL applications in this style.

The real challenge is a flexible dialogue, where the user can take over more and more of the control from the system, as he learns more about it. But when we lack the time and money to meet that challenge, we prefer to start out with the more helpful menu-oriented approach.

Nature of the task to be performed

Some computer-supported work is of an exploratory, unpredictable nature, where a computer application has no possibility of guiding the user on an assumed track; e.g. a data analysis system, with elaborate tools for retrieval, selection, and presentation of data, calls for a flexible dialogue with freedom for the user.

But in very many applications, there is a logical route on which to proceed; when there is a choice the user can be given a clear listing of his options. A routine task, such as data entry for an accounting system, is clearly suited for a dialogue with data input on fullscreen panels, and a high level of flexibility unnecessarily limits this.

The user's equipment.

With a slow typewriter terminal, the user does not want too much to be typed by the systems: He wants short explanations of menu options (or nothing at all); he does not want long print-outs just in case something interesting might be in them; and he wants to express his needs directly, instead of finding his way through a menu hierarchy. All this seems to suggest a command language to ease his work.

On the other hand, many users in this situation would probably most of all want to get other equipment, or faster communication lines. A command-oriented dialogue is often chosen, not really because the users' experience or the nature of the task so implies, but simply because menus (and data entry on fullscreen panels) are impractical with the current equipment.

In real life, you have to build applications for the real situation. In my company, we have several users with typewriter terminals, and we write many applications so that they too can use them. However, in most of the cases these users are waiting to get 3270-like terminals, to benefit from menu-oriented dialogue. There are only a few cases where small, portable asynchronous terminals are chosen on their own merits - such as bringing the terminal along to a customer, or bringing it home to do work there.

WHAT DOES THE DIALOGUE LOOK LIKE?

The dialogue style in an application depends on how certain features are designed:

How is data entry done?

When much data must be entered, a fullscreen panel is preferred. It prompts for all data related to one object within the application; or allows column-wise input for many objects at the same time. The simultaneous input of many items gives the user an overview; it lets him work faster; and it allows more intelligent data validation by the system.

On simple, asynchronous terminals such data entry is done via a sequence of prompts. A good dialogue will allow the experienced user to type-ahead answers to expected prompts, but the process is still tedious.

Command-oriented data entry can be preferred only when there is a commonly-used default for each item to be entered.

Example: The Swedish tax return form has a large number of fields to fill in, and most of them are zero (at least for me). Here, it could be a good idea to let the user start with an "all zero" situation, and let him change data items by commands like

```
TAXRATE 31.17 ; BANKDEPOSIT 5000 ; etc.
```

But there are few examples that really call for this kind of parameter + value input. Users like to see what values are used, and a fullscreen panel with the defaults visible allows that.

How does the user express his intentions?

This might be what most distinguishes menu-driven from command-oriented dialogue. There is a sliding scale, of course, maybe something like:

★ Selection by one integer 1,2,3,..., each representing one option. Or one character from a neutral set like A,B,C,...

★ Selection by a short code (1-2 characters),where the code is an abbreviation of something that the user understands. G = Get data and P = Print data; or GC = Get customer record and GA = Get accounting record.

★ Simple words to describe simple intentions: GET, PRINT, ERASE. These words (commands!) can be abbreviated.

★ A simple command language, e.g. as verb + object, or data item + VALUE. Verbs, object names, etc., can be abbreviated.

★ An elaborate command language, with many verbs, item names, values, all meshed into nested-parentheses.

★ A very rich syntactic structure...maybe APL itself.

★ (...there are a few steps more)

★ And finally, natural language. When supported.

Many users prefer the simple-syntax input, since they have too little time to learn an extensive command language. But note that all of the above are syntactical ways of expressing intentions, ranging from the very simple to the very elaborate. In a later section I will discuss some examples of non-syntactical user input.

How are the options presented to the user?

A typical menu-oriented dialogue displays the options with a one-line text explaining each. More information is available by a Help facility.

With an experienced user it is sufficient (and with a slow typewriter terminal often necessary) to keep explanations short. A list of the options, or of command verbs, is appropriate. In the strict command-oriented dialogue, the options are not listed at all, but can be seen only when the user requests help.

It is said that a command language encourages a flexible use of the system, because of the free input form. But often a user learns only a certain set of commands to perform a specific task, from a colleague or otherwise. He does not exactly know why the commands look as they do; probably he dares not experiment with the given parameters. He has no immediate reason to inquire what other commands there are, and is left ignorant of most features in the application.

When all options are clearly presented to the user, in a well-structured menu hierarchy, he is more apt to get curious and try out new parts of the system - parts that were included because somebody thought them helpful in his work.

Which options are available?

Many applications are built so that in a given situation only certain options are available. This can be described as a hierarchy, in which successive choices are made. Typical menu-oriented systems have this organisation, as opposed to many command-oriented applications, with a "flat" hierarchy, permitting most commands in a main dialogue loop.

Now, as the user gains experience with a menu system, he will feel uncomfortable with the control implied by the hierarchy. In many such systems he will discover that the control is not there. Successful menu-oriented applications allow the user to take over more and more control, in various manners:

★ Direct selection within the hierarchy: a user working within alternative 3.2 can enter something like "GOFAST 2.8".

★ A temporary excursion into one operation, after which the user returns to the previous situation. The help facility is an important, but maybe not typical example of this. An extreme case is when some applications allow the command "APL......", where the ..... should be executed as an APL expression.

★ Starting a parallel session, such as splitting the screen layout into two. At times, this is supported not by the application, but by the time-sharing system, or by the user's terminal.

★ Letting more options (commands) be available than those listed on the menu. The listing is the system's best estimate of how to proceed; it is displayed for pedagogical reasons, and not because of the logic of the application.

★ Allowing selection codes (commands) to include parameters, even if that is not stated on the menu display. Say the user normally enters GET (or G) to get data; he expects the system to prompt him for year and month. But the experienced user knows that he can at once enter GET 8312.

So there are several ways to let the user grow into a more flexible use of the system. I feel, however, that the right starting point is one where the user gets much information and help, where his inputs are simple, and where the dialogue guides him on a suitable track through the application.

EXAMPLES OF NON-SYNTACTICAL INPUT

How does the user express his intentions? In the typical menu selection situation, there is one more way: he can point into the menu! (Whether he uses a light pen, the cursor, or his finger is of less interest.) There are many situations where the user's work is made easier if the dialogue can utilize the geometry of the screen instead of an intricate syntax. I would like to elaborate a bit on this theme, which I think fits in well with the helpful, menu-oriented dialogue style.

Example: Consider text editing; and assume the user decides that a certain paragraph should be moved. Syntactically, this is expressed with something like

MOVE 135: 140 231

—with difficulties in remembering which are the "to" and "from" line numbers. An alternative is to let the user mark the paragraph to move (with characters in a special marker column), and similarly the destination. Here the user expresses what he means in a direct fashion, closely modelling the operation to be performed.

Note that the line numbers in this example are artificial — the text itself does not need them. They are added by the "syntactical" editor just to make the syntactical expression possible. In a non-syntactical editor they are not needed.

Another example with text editing is line split: A syntactic way to request it would include line number, plus some indication of where the split should be done. Very intricate! Instead, let the user point with the cursor to indicate where the split should be, and nothing can be simpler.

Example: Assume a tool to do yield calculations on bonds (common in financial institutions). The user enters a set of base data for the bond in question, and a suggested price, to obtain the yield (which is his key factor). Alternatively, he can enter a stipulated yield to get the implied price; in fact he wants to go back and forth between these two modes of calculation. Well, let him do that on a fullscreen panel:

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                                          *
* BASE DATA         ________________________________________              *
* . . . . . .       __________________________________________            *
* . . . . . .       ____________________________________________          *
*                                                                          *
*        PRICE    ______________________      YIELD ______________         *
*                                                                          *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

In which direction should the calculation be performed? The user will not have to explicitly (syntactically) state the direction. Instead the application can use non-syntactic information that the user supplies — this time without even thinking about it. It can detect which of the fields PRICE and YIELD has been changed, or draw some conclusion from the position of the cursor, and take the correct action.

Consider also data entry on a fullscreen panel:

Example: Assume that the command INDATA leads to the following panel:

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                                          *
*                                                                          *
* NAME              ______________________________________                *
* ADDRESS           ____________________________________________          *
* AMOUNT            ____________________________                          *
*                                                                          *
*                                                                          *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

If you prefer a syntactical model of this, you say that the panel lets the user type ahead three answers into the input buffer. Conversely, assume that before he sees the panel the user types:

INDATA;BJORN BORG;MONTE CARLO;9999999

With a syntactical description, you expect the system to look into the input buffer for name, address and amount. Most applications don't; the panel is considered to reside outside the syntax of the dialogue. Instead, field inputs are assigned to data items depending on panel position. In the following section are discussed some implications of regarding the data entry as being non-syntactical (existing outside the input buffer).

So, the observation in this section is that not all user input can be regarded as syntactical (and thus possible to pre-type into an actual or imaginary input buffer). Successive menu choice can fit into a syntactical description, but when the dialogue utilizes the geometry of the screen, or a light pen, or colour, the input is clearly non-syntactical.

And the argument made is that these other kinds of input can give the user a higher productivity. Many operations are more easily described in other ways than with a syntax.

RERUNNING THE DIALOGUE — AND PRERUNNING IT.

One question about good dialogue design is: Good for whom? For the user, of course. But other people are interested, as well:

★ The ultimate end user, waiting on the telephone while the terminal operator is struggling with the system.

★ The programmer, overloaded with work, and strongly in favour of standard program utilities that ease his work, while they unify systems for the user.

★ The production staff, concerned about CPU load and communication lines.

★ The company auditors.

Maybe to your surprise, I have a few words about the auditors. Two of their (very legitimate) concerns are:

★ They don't want people to have freedom to do things that they are not allowed to do. However, a competent user can always try to do things in free APL, that a controlled dialogue won't allow him. So this is not really an argument for a strictly controlled dialogue.

★ They want to be able to get a transcript of terminal sessions, providing the opportunity to reconstruct critical operations.

It is obviously difficult to provide a transcript of a session with much non-syntactical input. "What did the user really do; where was the cursor positioned when he hit the Enter key?" And similarly, it can be impractical to provide a transcript when data entry is done on a fullscreen panel. The session log of a command-driven application is what the auditors need.

In fact, not only the auditor but also the user himself is interested in the transcript. With a menu-driven application, you find yourself asking questions like: "How did I produce this nice graph? Well, first in menu 6.1 I did this, and then...". A related question is that a command language allows for easy introduction of command procedures, where the user can store away sequences of commands for later use. Very seldom, if ever, can fullscreen, menu-based systems utilize command procedures. Normally that is possible only if the application also supports a command-oriented dialogue.

So I admit that in these respects (auditing, the session log, and command procedures), the menu approach is weaker than a command language. This holds even stronger with a non-syntactical dialogue design.

CONCLUSION

I will not claim that a menu-oriented dialogue is the best choice in every case — I have myself discussed when its drawbacks get noticeable. Each application must be judged by itself; but in my company we often use a fullscreen, menu-based design for the following reasons:

★ User productivity benefits from the more helpful menu dialogue design, mildly controlling the user's route through the application. This is true especially for the many casual or infrequent users in the company.

★ Most applications are of a rather uncomplicated nature, and users prefer simple inputs to describe simple operations.

★ The users appreciate that APL and IMS application have a similar dialogue style.

★ Data entry on fullscreen panels is superior to line-by-line input.

★ Listing alternatives on an explicit menu encourages users to learn more about an application.

★ The menu-oriented approach is preferred as a starting point. Ambitious applications can arrange the dialogue to let the user gradually assume control.

Everybody agrees on the advantages of graphical, diagrammatic output. I have pointed out that also for user input, it is valuable to use the geometry of the screen, and not always rely on a syntactical mode of expression. This does not necessarily imply very sophisticated terminal equipment; much can be done with good screen layouts, using only character input and the cursor's position. These non-syntactical input schemes are a natural extension to the menu-oriented approach.

Gösta Olavi
Senior Systems Analyst
Dept. for Marketing & EDP
S.E. Banken
Stockholm
Sweden.

Sharp APL
APL PROFESSIONALS

I.P. Sharp Associates is a highly successful computer services organisation operating worldwide through 60 offices in 20 countries. The group specialises in the APL language, with packages and applications covering a wide range of industries, especially international banking and energy.

Vacancies exist in our London and Dublin offices at senior and junior levels for those with APL experience. Salaries are competitive and unlikely to be a problem for the right candidates.

Please write in the first instance, with brief career details, to J.D.H. Weatherby, U.K. Manager at the address below:-

I.P. SHARP ASSOCIATES LIMITED

European Headquarters
Heron House
10 Dean Farrar Street
LONDON SW1H 0DX
ENGLAND

Tel: 01-222-7033

I.P. Sharp Associates

TOWARDS A TEACHABLE INTERFACE

by Adrian Smith

Introduction

This paper was given as my contribution to the panel on 'Menus versus Command Languages' at the 1984 I.P. Sharp Users Meeting. It is an attempt to view the whole user-interface debate from a different angle, and concentrates on the ability of a system to learn rather than on the precise style of the dialogue. As systems move deeper into the role of management support, I feel that 'teachability' will become increasingly important. There are two main reasons for this:

— Usage is optional. A management support system (MSS) must appear initially attractive, but must not frustrate the expert with tedious or inflexible dialogue. It will soon lapse into disuse otherwise.

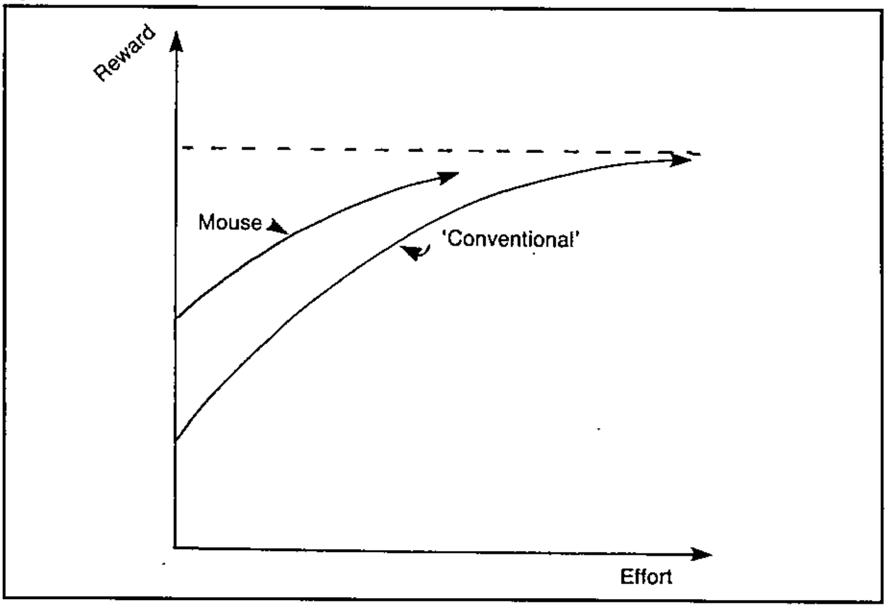
— The application is unpredictable. No-one knows in advance how a particular MSS will be used by a particular manager; any attempt to pre-define 'handy' shortcuts is doomed to failure. Such paths can only evolve from the two-way interaction between each individual user and the system.

The next section looks at menus and 'masters' (command languages) in this context. I hope it sheds some light on readers' own experience of the many types of user-interface they will have met. Having pinpointed the weaknesses of both styles, I then want to show how the idea of teachability overcomes many of these. Finally, the paper outlines a simple idea that lets you turn almost any APL system into one that can learn from its users.

Reward per Unit Effort: a Measure of Attractiveness.

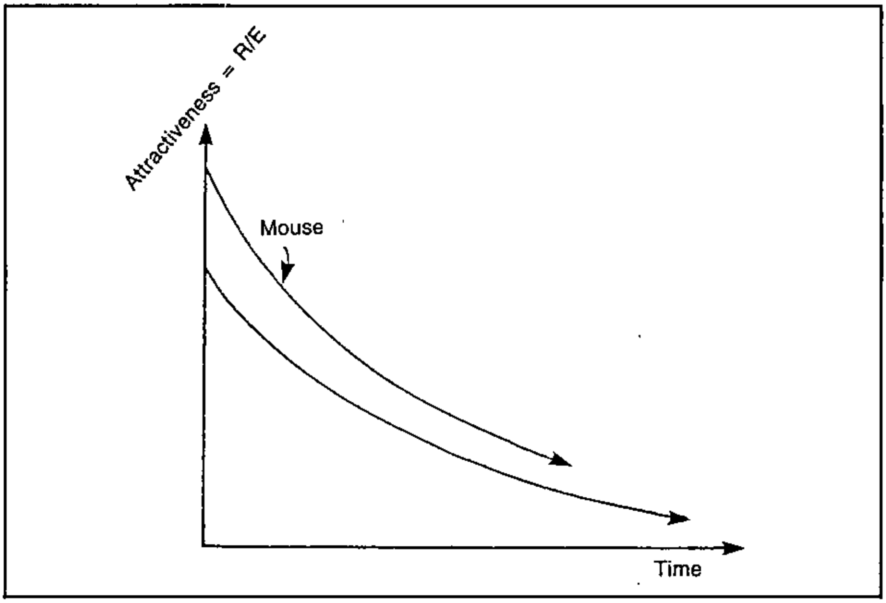
When you ask someone how much they like a system, the answer almost invariably depends on the ease with which they get results. Taken over time, one can plot the reward obtained against the effort expended; for menu-driven systems the graph comes out like Fig. 1:

Fig. 1



The 'first time' or occasional user gets a reasonable reward for virtually no effort at all; the system is initially very attractive (Fig. 2):

Fig. 2



Unfortunately the finite range and segmented structure lead rapidly to diminishing returns, until the user reaches a dead end where all the options are known and no increase in effort brings any new reward. At this point the attractiveness of the system has fallen close to zero.

The opposite profile is typical of many command-driven systems. For the really archaic examples (e.g. the TSO operating system) the R/E chart shows an extreme case of unattractiveness (Fig. 3):

More representative are the 'English-like' commands of most enquiry systems and the 'Fourth Generation'. They take a bit of learning, but have the great merit of being effectively open-ended; more effort yields an increasingly greater reward. Fig. 4 shows the attractiveness profile of this approach.

The big snag is that first encounter. If you don't know the right words you will get precisely nowhere, and you may never reach the point where your interest becomes self-sustaining. It usually takes some clearly perceived benefit (well known in advance) to get users through that initial period. Before I move on to describe an approach to the user-interface which attacks many of the above deficiencies, I want to spend a couple of paragraphs on two current ideas which I feel are red herrings.

Fig.3

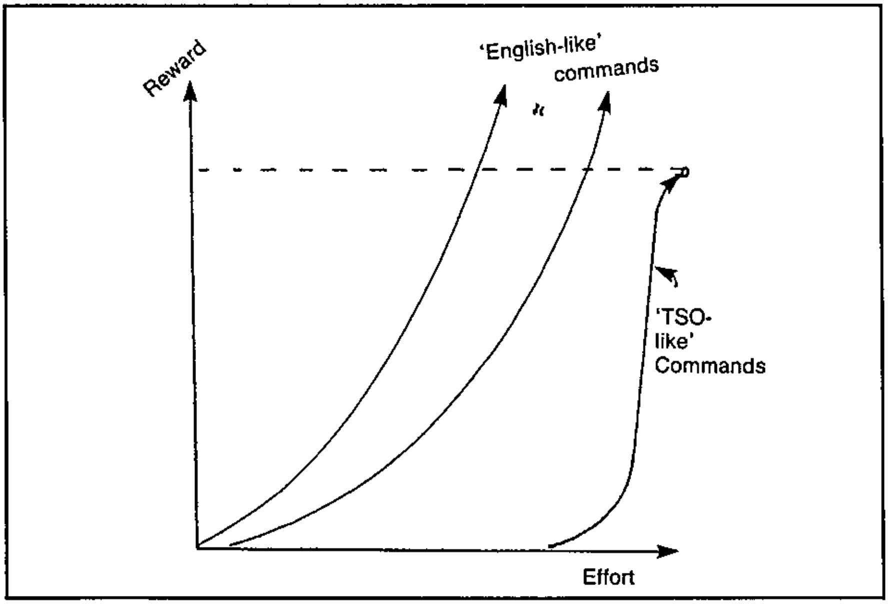
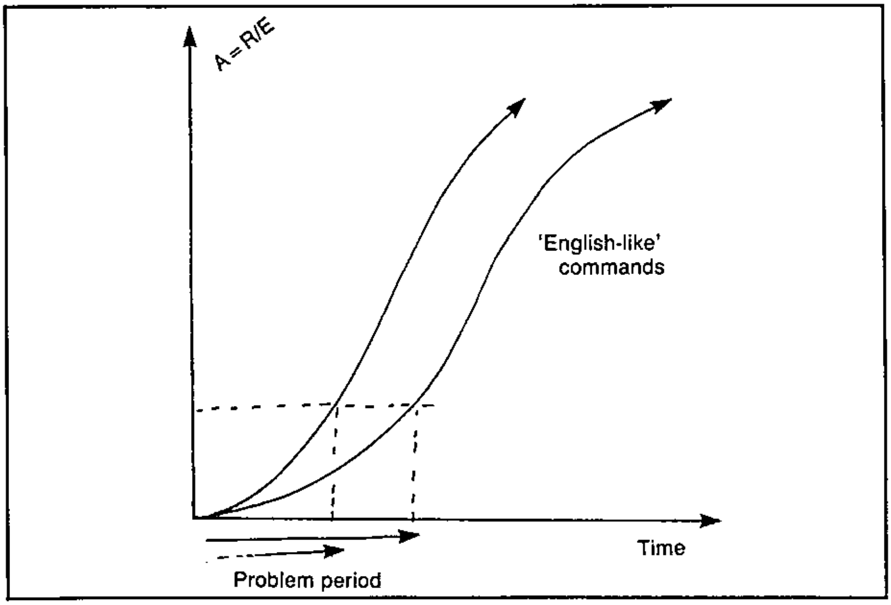

Reward
'English-like'
commands
"
'TSO-
like'
Commands
Effort

Fig. 4


A=R/E
'English-like'
commands
Time
Problem period

Two Red Herrings

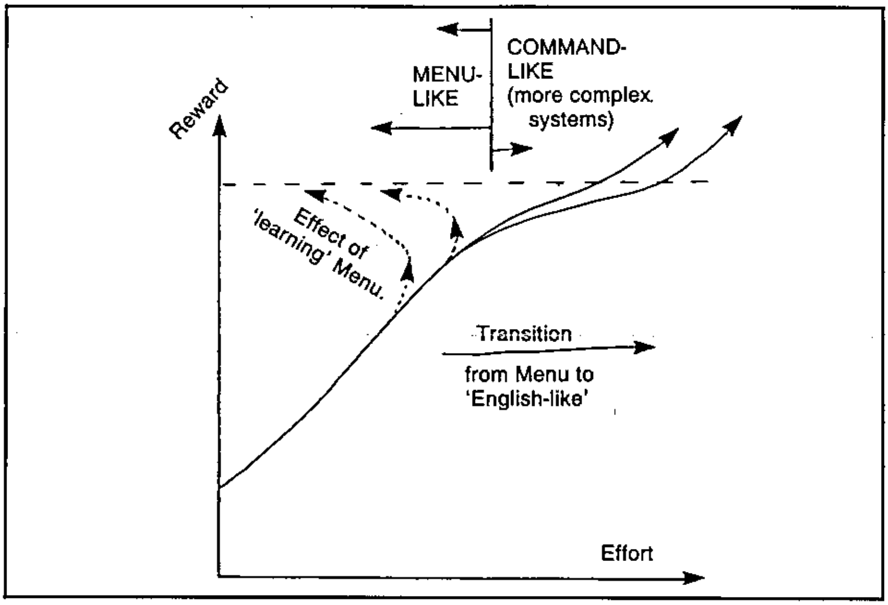
It is quite possible (in fact very easy) to make the default menu option 'whatever you did last'. In really trivial dialogues (the 'VisiWord' spelling checker is a good example) this can be very helpful. Just imagine being able to ask your autobank for £20 by entering an id number, and hitting a 'ditto' key. The problem with extending this idea to the MSS is that it copes very badly with complex multi-level menus. It is the menu-structure that gets in the way; clever tricks within each menu are no help at all!

The second red herring is 'natural language'. English is wordy, imprecise and context-sensitive. As long as communication is via keyboards and screens, the minimization of typing will be more important than the ability to waffle and circumlocute. Even given a workable device for voice-input, I find it probable that a computer faced with the stricture 'Dogs must be Carried' (see any London Transport escalator) would feel obliged to obtain a suitable dog before venturing into the Underground. So much for English!

How can an Adaptable Interface Help?

In simple terms, by giving users a Reward/Effort profile similar to that shown in Fig 5:

Fig.5



The initial attractiveness is high, because the system can be preset with the 'obvious' commands which any novice user is likely to need, for example:

```
Please point the cursor at any of the following lines
and press (ENTER) ......

CHANGE DATA           ...   update the current sales figures
CHANGE BUDGETS        ...   alter the latest estimates
DESCRIBE              ...   for full details of the system

)SAVE         ...   to take a safe copy of your data
BYE           ...   save the data as above, and sign off

____________________________________________________________
```

At this stage it behaves just like a menu; for 'one-time' or very occasional use the two are indistinguishable. The difference only becomes apparent when someone becomes confident enough to try out commands on his own. This vital step is made much easier by the presence of numerous examples ripe for adaptation:

```
SHOW DATA WHERE CODE IS 'KK'
```

... is easily turned into:

```
PRINT DATA WHERE DEPT IS 'MKTNG,SALES'
```

... and then into:

```
PRINT 'NAME,AGE,SALARY' WHERE (AGE > 25)
        AND (SALARY < 10000)
```

... and so on. Some effort is indeed required, but it is far easier for a user to re-work an initial skeleton than it is to invent the whole of a complex query out of thin air.

So far the adaptive idea has yielded the initial attractiveness of menus together with the encouraging open-endedness of a good command language. There is one additional bonus: once a user has taught the system his favourite commands they stay where he put them for next time. The effort needed to get results actually decreases as time goes on. Even after a lengthy break a user will be able to pick up the words and syntax very quickly; the options offered are always the last thing he did!

Next question: can it be done? Oddly enough it can, and very cheaply too. The following section outlines an extremely haphazard path towards the retrospectively obvious.

Steps Towards a Teachable System

The story begins back in the early days of interactive computing, when APL meant slow teletypes, and none of us knew the first thing about dialogue design. We instinctively avoided menus and lacked the programming skill to build command interpreters; this left only one real option - APL was left to do the work. The user (having achieved the near impossible feat of dialling up and logging on) simply loaded an APL workspace and was let loose in a totally open system with commands like 'GO', 'SCHEDULE', 'EDIT' at his disposal.

Old habits die hard, and even in the modern world of full-screen entry many of us have stuck to this seemingly archaic style while our colleagues moved on to colourful menu panels and function keys. Oddly enough the result has been the development (largely by accident) of a totally flexible interface, which looks ideally adapted to the next leap forward - touch-screens, mice and the rest.

First came some very basic attempts to make the raw APL environment more forgiving. The system insists on upper-case; most users get sick of holding down the shift key, so why not trap their commands and pass them on in big letters. While we are about it we might as well check for any unpaired quotes and have a look to see if all the words actually exist in the workspace. The odd SYNTAX ERROR will still slip through, but the vast majority of simple typing mistakes can be picked up. There is however one big snag with this dialogue: it assumes that the user can type. All very well when the functions are limited to short single words, but when people start getting ambitious:

```
SHOW STAFF WHERE DEPT CONTAINS 'PRODUCTION'
SORT STAFF BY 'SERVICE' & PRINT ALL DATA
```

... the effort involved is only repayed when the rewards are quite significant. It is particularly frustrating when a user has entered some very complex expression, and has to retype the whole thing because of one trivial mistake. Fortunately IBM (in our case with a new release of VSPC) offered a way out.

Step 1: A Short-term Memory

You still type things in along the bottom line of the screen (echos of the teletype days again), but anything already entered is displayed above (or on a previous screen having scrolled out of sight). To get it back you simply move the cursor up to the required line and hit the (ENTER) key. This brings the whole line back to the bottom to be re-executed or modified as required. Not only does this apply to your own commands; you can also retrieve and execute text which has been put there by the computer. With this simple step it suddenly became possible to write a 'HELP' command which cleared the screen and displayed something like this:

```
******************************************************************
*                                                                *
*  CHANGE DATA              ...   to update the figures          *
*  SHOW DATA                ...   for a simple display           *
*                                                                *
*  SORT DATA BY 'xxxxxx,xxxxxx'         ...   change the order   *
*  SELECT DATA WHERE DEPT IS 'xxxxxx'   ...   pick a subset      *
*  HELP                                 ...   get this back      *
*  DESCRIBE  ...   for full details                              *
*                                                                *
*  )SAVE     ...   to take a safe copy of your data              *
*                                                                *
*  ____________________________________________________________  *
*                                                                *
******************************************************************
```

Many of the commonest expressions can be pulled down from this screen and executed with no modification. (Anything to the right of two dots is a comment). Of course this is still some way from a truly teachable system, but at least the code of the 'HELP' function is so trivial that it can be redone at a moment's notice as a pattern of usage emerges. Of course the system does learn quite well in the short term; users can go back up to 10 screens, so even rather complex commands can be built up in easy stages. The final version often gets noted in a 'little black book' for future reference and possible inclusion in 'HELP'.

There now followed a brief excursion up an attractive blind alley. Why not include a command like 'CHANGE MEMO' to let people alter the help information themselves? It sounds obvious and straightforward, but (to the best of my knowledge) no-one ever took advantage of it. Perhaps the effort of re-typing the command (and indeed of remembering it) was too great. For several years our systems remained fully teachable only by the programmers, and many became increasingly hard-wired as the options multiplied and menu-trees began to take over.

Step 2: Long-term Learning

The final development not only sounds straightforward and obvious - it was. As always with such things you wonder why it took you so long to think of it. The basic principle is to split the screen up like this:

```
*******************************************************************
*                                                                 *
*  Some Kind of Heading                         Date and          *
*  --------------------                         Time of Day       *
*                                                                 *
*  ............................................................   *
*  .                                                          .   *
*  .                                                          .   *
*  .                                                          .   *
*  .  Area where instructions are entered .....               .   *
*  .                                                          .   *
*  .                                                          .   *
*  ............................................................   *
*                                                                 *
*  ............................................................   *
*  .                                                          .   *
*  .  Area where APL displays results and messages ...        .   *
*  .                                                          .   *
*  ............................................................   *
*                                                                 *
*  ------------------------------------------------------------   *
*                                                                 *
*     Notes on what the PFkeys do ..............                  *
*******************************************************************
```

Typically there will be function keys for 'Leave this Menu', 'Hardcopy' and 'Help'; everything else (including transfer to other menus) is done by commands. So far nothing new. What is novel (I think) is that the area where commands are entered is saved along with the rest of a user's data, so that each time he calls up the system the first thing he sees is whatever he did last! As I said before, the principle is very simple. All you need is a driver function which uses standard full-screen facilities to format the screen, and writes the current version of the menu into the middle bit:

USEMENU 'REPORTS,HELPREP' ... names of menu and 'help' panels

This driver waits for (ENTER), reads back the menu, and then has a look to see where the cursor was; it takes everything from the beginning of that line (or from the previous colon) up to the end of the line, or to the next colon, whichever comes first. An exception is made for ampersands which can be used to chain a series of commands together:

SELECT DATA WHERE DEPT IS 'SALES' &
SORT DATA BY 'AGE,SERVICE' & PRINT 'NAME,SALARY'

A certain amount of laundering can be done en route, for example a comment facility is obviously essential, and it makes sense to close off unpaired quotes:

Load 'staffdata ... personnel details
Lib .... your library : BYE ... save and sign off

Having checked that all the (unquoted) names really are lying around in the workspace all the driver needs to do is bang an (EXECUTE) in front of each part of the command list, and loop through them from left to right. As long as all the top-level functions return explicit results (even if only a token such as 'READY'), it can then pick these up and output them (formatted of course) into the lower window on the screen.

In practice I find that it pays to re-assign the contents of the 'action' field back into the menu before I try to execute anything. This way, if there is a disaster, the user can at least restart by correcting the line which failed. Of course those readers lucky enough to use an APL with error-trapping can catch this kind of thing automatically, and thus make the whole procedure rather safer. In fact the only real snag is system commands; these have to be shoved into an input stack, along with a restart to the top-level menu to get things going again. This means that you 'lose your place' in the tree whenever you are conscientious and ask for a ')SAVE'; an annoyance I haven't yet found any way round!

Example and Conclusion

A typical 'Reports Menu' (copied straight out of a real system) looks like this:

```
****************************************************************
*                                                              *
*  Show data : Show rows unless appl contains 'C,V'            *
*  Show rows where serial contains 'ad923'                     *
*  pagebreak on type [;,1]                                     *
*  Show    rows where (ADDRESS IS 'LU78IAA'):                  *
*  select rows where LOCATION contains 'kit'                   *
*  show rows where appl contains 'T' unless sf contains 'P'    *
*  Select rows where chosen and SF contains 'M'                *
*                                                              *
*                                                              *
*  Select cols 'ADDRESS,APPL,CICS,SERIAL,TYPE,LOCATION'        *
*  Select all rows : Select all defn : SORT ROWS               *
*  PRINT DATA : SAVE :SETUP :BYE                               *
*                                                              *
*  ----------------------------------------------------------  *
*        PF 9 : Finish      PF 1 : Hardcopy      PF 12 : Help  *
****************************************************************
```

As you might expect, it doesn't take long for the keen user to get that nice default menu into a right royal mess! Much as this may offend our professional pride, I have one thing to say in my defence of this intrinsically anarchic approach to the user-interface: it works! On that vital yardstick of Reward/Effort it is initially attractive (I believe it was Al Rose who first used the term 'Point-and-Grunt' to describe this style of dialogue); and yet at the same time it is totally open-ended. As long as APL can execute it, then anything goes.

Finally, even the menu structure is fluid: because the menus themselves are simply repeated invocations of the same driver with different panels, the whole thing is naturally recursive, and it doesn't give a damn whether you invoke 'Reports Menu' from 'Graphics Menu' or vice versa. Anyway, I always did like open systems; perhaps the whole thing is just another way of justifying an old prejudice? Try it and see!

CASE STUDY

Notes by Adrian Smith

The paper that follows actually came from the Association meeting on 'APL and Knowledge-based Systems' in September. However on reading it I quickly realised that it fell very nicely into the Case Studies category.

It covers the development of a system called PATTIE, which is all about pipes. Given the need to transport (say) hot chocolate half a mile in freezing weather, what sort of pipe do you need? Should it be buried or lagged? Is it a serious industrial hazard to the surrounding community? Can it indeed be done at all??

The experiences of the author in writing PATTIE 'from the ground up' make fascinating reading, and I hope you will find his paper as interesting and informative as I did.

in this issue of VECTOR we include the outline and project background of PATTIE.

- VECTOR 4 will include the remainder of the paper, which covers problem areas, and notes on the implementation.
- for convenience, the references will be printed in both issues.

Current and Future Technologies

Mitchell Watson (IBM)

The speaker is Vice-president of the Systems Products Division, so his words were followed avidly by the professional IBM-watchers present. Unfortunately the rest of us rapidly became bamboozled by a string of acronyms and serial numbers, and it is rather hard for me to report anything useful here.

I believed him to indicate that the emphasis would be increasingly on the world of the PC, but that it would gradually acquire all those letters like SNA and VSAM that have long been familiar on big machines. Basically IBM can't afford to write off an architecture that has grown up in the world of dumb terminals, so even though the handling of text/graphics/voice will move out to the workstation, old favourites like VM will still matter internally. Interestingly, he was quite clear that the 5-inch floppy is here to stay, although it will soon be complemented by the miniature optical disk. I wonder how supporters of machines like Apricot and Macintosh feel about that?

User Productivity Tools

Bob Metzger (IPSA)

This was one of those sessions which worked far better in the printed version than in the actual talk. In fact I read the paper in the pub (they do rather good draught Bass in the 'Duke of Richmond') over lunch, and I found this much more rewarding than listening to Bob's lecture! Accordingly I am going to forget my rather meagre notes - which clearly fail to do justice to 2 hours of material - and simply attempt to precis the published paper. If you can get hold of a copy of the original, I would strongly recommend it for detailed study.

Bob suggested that two elements of any 'User-productivity aid' need careful scrutiny. Firstly it must provide genuinely integrated data, and secondly there must be the possibility of non-procedural processing. The first of these needs little explanation, but the second leads to four further questions which are fundamental to any evaluation.

- Is the processing 'reactive' or 'proactive'?
- Is the user's view 'segmented' or 'apertural'?
- How is 'natural language' handled?
- What about graphics support?

By 'reactive' Bob essentially meant menus. The user's options are limited to selection, asking for help, and backing out. 'Proactive' systems are typified by spreadsheets; in Lotus-123 the user can:

- type data into a cell
- type a mnemonic command
- use the cursor and function keys.

In choosing the domain that we did we were looking for an area in which the knowledge, or expertise, could be 'codified' in the form of rules, relatively simply and straightforwardly, due to the time constraint involved (the system had to be finished inside six months).

Thus the idea was formally stated as: the domain would be that of Pipe Analysis related to Mechanical Engineering. A first-year undergraduate would enter into the system details of the proposed pipe site, the environment, the transport medium flowing through the pipe, etc. and the system would return aspects such as: what pipe materials could be used, how could/should the pipe be jointed and fixed to its surroundings and what other considerations must be taken into account (e.g. lag the pipe, bury it, etc.).

Basic Operation

We had identified what seemed like a reasonable domain, therefore work could start on the system design. At this stage we knew effectively nothing about the design and development of Expert Systems, although we did know what they could do. From the point of view of a naive entrant to the field, there was little documented that could help with the question "How do you develop an Expert System?".

As mentioned earlier, there was not a great deal of time in which to write the system. Therefore there was, relatively, little time for doing a vast amount of background reading and research. It was decided to use a well-documented system as a guide on which to base our Expert System.

The system chosen was the medical diagnosis system, MYCIN and its associated knowledge acquisition facility, TEIRSIAS (ref:1).

MYCIN was based on the concept of production rules. An example of a MYCIN rule is:
  IF the infection is PRIMARY-BACTERENIA
  AND the site of the culture is one of the STERILE SITES
  AND the suspected portal of entry of the organism is the gastro-intestinal tract
  THEN there is evidence that the identity of the organism of BACTERODIES.

Rules can therefore be seen to be a means of writing down, and storing, knowledge. It will be seen later how these rules are used by the system.

The next stage was to develop some rules, i.e., codified forms of knowledge, or expertise, involved in our area of Pipe Analysis. Examples of the original rules are:

RULE 1
  IF the transport medium is resistant to cold
  AND if it has an absence of taste
  THEN polyethylene can be used for the pipe.

RULE 2
  IF the pipe diameter is greater than 1.2 m
  AND the pipe is made of polyethylene
  THEN a flange joint is required

RULE 3
  IF the pipe is on the land
  AND it is near a fire hazard
  THEN it must be buried

The format of these rules is interesting because it is a domain expert's idea, without the aid of a Knowledge Engineer of how he would formulate, or describe, knowledge of his domain. Knowledge Engineering is a very important part of the design process more of which will be mentioned later.

The next problem was the most difficult and yet the most interesting. How could the knowledge be encoded such that it would be in a format that the Inference Engine could work upon it?

After some deliberation it was decided as a first stage that the various details that had come through by the rules provided would need to be grouped under certain classifications where things could be logically divided up. For example; around the concept of "pipe site" there were things such as: buried, buried in a trench, above the ground, along a wall, etc. There were various types of pipe material: if the transport medium was an acid then do not use polyethylene, or if the purpose of the transport medium was for food manufacture then do not use, for example, brass pipes.

Given classifications such as these and the thirty or so rules provided, the following list of classifications evolved: pipe material, pipe environment, pipe property, pipe site, pipe diameter, pipe joints, pipe fixings, transport medium material, transport medium property, transport medium purpose and notes.

From the rules, examples of classification attributes are:

Pipe environment — near fire hazard, near road, outdoors, unshaded, still water, running water
Pipe material — polyethylene, cast iron, plastic, aluminium
Transport medium property — resistant to cold, absence of taste, fluid, hazardous, hot
Transport medium material — hydrofluorous acid, drain water, sulphur trioxide, chlorine, milk

The original rules could now be rewritten into a format based upon this concept. For example the three rules above became:

```
RULE 1
  IF a transport medium property is resistant to cold
  AND a transport medium property is absence of taste
  THEN a use-type is polyethylene

RULE 2
  IF the pipe diameter is greater than 1.2 m
  AND the pipe material is polyethylene
  THEN the pipe joints are flange

RULE 3
  IF the pipe site is on land
  AND the pipe environment is near fire hazard
  THEN the pipe site is buried
```

The basic operation which then followed was:

User gives: the transport medium flowing through the pipe

System returns: Possible pipe material types
How to joint each type
How to affix each type
'Properties' of each type
Notes associated with rules

Language and System Choice

PATTIE was implemented on a 68000-based multi-user micro system, it was written in C and ran under UNIX System III. The reasons for using C were:

i) its portability. If the system was going to be used by the Mechanical Engineering Department, it would be on a different machine. C was the most portable language available.

ii) many commercial Expert Systems are written in Pascal or C. Doing a similar system would give an insight into why this is and the ease of doing it.

iii) experience of C and UNIX was, in itself, desirable.

Prolog would probably have been a more suitable language to use because:

i) it has its own in-built pattern matcher

ii) it has its own in-built inferencing mechanism (using depth-first search techniques)

iii) it would have allowed us to concentrate on developing the Knowledge Base and the Natural Language Interface.

However, the reasons why C was chosen in preference to Prolog were:

i) it was considered that more would be learnt by writing our own inferencing mechanism rather than using the mechanism which is in-built within Prolog

ii) our Prolog interpreter did not work correctly!

PATTIE SYSTEM OPERATION

General

How does the system actually manipulate the knowledge to produce 'results' from the program?

So as to concentrate efforts, the system works in terms of certain types of pipe material. The Knowledge Base is 'partitioned' into these types (for the purposes of this discussion let us say brass, concrete and polyethylene). There is also a partition for 'general' rules, i.e., rules relevant to pipes in general and not to any specific pipe material type. Rule 3 is a general rule.

New rules are entered into one of these partitions, although new partitions can easily be created.

The system, given the transport medium, tries to infer everything it can from the rules it has, asking questions of the user when it "gets stuck".

As mentioned, the Knowledge Base is split into partitions, one for each type of pipe material. Given the transport medium, the system will initially try to rule-in or rule-out any of these pipe types. To do this it uses a further partition of rules called "Use-Type" rules. An example would be Rule 1, above. This example says that (given that the two conditions are *both* true then) it is possible to use a polythylene pipe. A Use-Type rule can also rule-out a pipe type (i.e. *not* Use-Type).

This is a very useful feature because it means that a whole section of the Knowledge Base can be eliminated from any future processing. This saves a considerable amount of time and effort from the point of view of the system file reading and pattern matching.

In the case where it is not possible to rule-in or rule-out a pipe type, the system will go ahead and assume that to use the pipe material would be reasonable.

After completion of the above stage in the processing, PATTIE will continue working to try to discover as much about each pipe type that has not been ruled out. When the searching of those relevant partitions of the Knowledge Base has finished, and all possible discoveries have been made, all the items of interest (mentioned earlier) are returned to the user.

Everything discovered by PATTIE comes from comparing rules in the Knowledge Base with other rules (this is detailed in a later section; the Inference Engine). When PATTIE cannot find a rule that will help to satisfy an inference being performed, as a last resort the user will be asked. All answers to queries are stored by the system in what is termed the Short Term Memory, or STM for short. Even answers where the user said NO are stored. This is on the principle that should it become necessary to ask the same question again, the STM can be checked first rather than bother the user who will not understand why the system appears to be repeating itself!

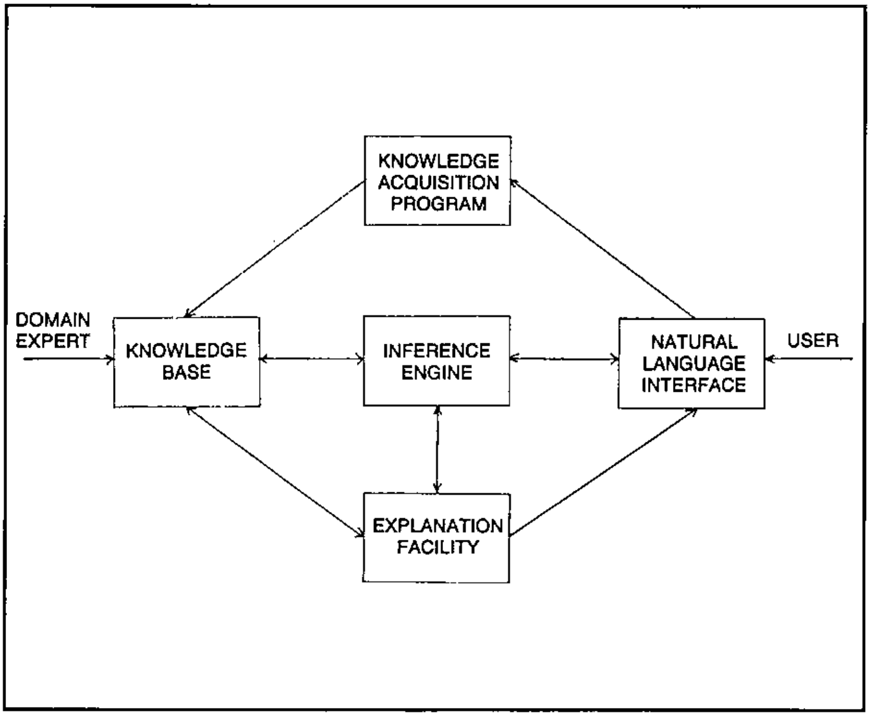
Also stored in the STM are 'intermediate' results discovered during processing. This saves work should the system re-trace steps already covered.

An important facility of PATTIE is the DON'T KNOW option. When the system has to ask the user a question there is the possibility that he will not know the answer. Therefore, to avoid the user having to give a potentially misleading piece of information to the system, he can reply DON'T KNOW to a question. The system must take this to mean a reply of NO, in that it proceeds no further with its current line of reasoning (indeed, it *cannot* proceed further) but, at the end of processing, the user is reminded of all questions which were answered in this manner, so that results can be put into perspective.

Another important option offered to the user was that of having a hardcopy at the end of the session detailing everything that happened during that session. This is necessary when a VDU was being used.

It is generally accepted that an Expert System should consist of the five components shown in Diagram 1. Each of these areas will now be gone into in more detail.

Diagram 1



Knowledge Base

The core of the system is the Inference Engine and the Knowledge Base.

The Knowledge Base contains rules about the subject domain. To infer (discover) anything, the system has to be able to compare rules with other rules. Therefore, there has to be a 'standard' to which these rules must conform.

A rule clause takes the form:
  type — classification — associated text

Examples:

```
IF    transport-medium-property   hazardous
AND  pipe-environment   hot
THEN pipe-property   lagged
```

Firstly, what are 'classifications'? These could also be called 'attributes'. Information about the domain is held with respect to these classifications. They are, if you like, a point of reference that PATTIE can use to compare a clause of one rule with another. Any new rule must be defined in terms of these classifications. A list of classifications and examples were given earlier.

Secondly, what is the "associated text"? Well, the classifications are fixed, rigid entities but the details associated with a classification in a clause of a rule are "free format" (i.e. you can type in whatever you like). It is this part of each clause that will be used during the "pattern matching" comparisons of the Inference Engine.

The example clauses just shown are not very 'readable'. Therefore the idea of 'noise' words is used. Noise words recognised by the system are THE, A, IS and ARE. These words can be inserted into the above rule to give:

IF a transport medium property is hazardous
AND the pipe environment is hot
THEN a pipe property is lagged

These words are ignored during any processing involving the rule.

To save storage and access time the rules are stored in a compact form. Each of the 'types' (IF, AND, THEN, etc.) and each of the 'classifications' (pipe site, pipe property, etc.) are replaced by single values. Only the "associated text" needs to be stored in its full form, although even this is stripped of any unnecessary spaces. Upon retrieval of a rule from the Knowledge Base, should the rule need to be printed out, conversion to a more English-like format takes place (as shown above). Great use is made of these noise words.

Also available but as yet unmentioned, is the ability to add a purely textual note to the end of a rule. This was found to be a very useful facility. An example would be:

IF the pipe diameter is large
THEN the pipe fixings are concrete ballast
NB in this rule a protective material interface such as flux or plastic foil is required. sdd/11-83

The storage required by this rule is increased dramatically but the advantages are worth it.

As stated earlier, the Knowledge Base itself is divided up into logical 'partitions'. The first partition is for Use-Type rules (rules which suggest whether, or not, certain pipe materials can be used for the scenario in question). The second is for 'general' rules (rules which are common to all pipe material types). Subsequent partitions hold rules specific to pipe materials e.g., there could be a partition containing rules about polyethylene, one containing rules about concrete, one about brass, etc.

The ability to group rules logically is very handy; it prevents duplication of effort and allows flexibility as well as the processing savings already stated. Unfortunately, it complicates the user view of the Knowledge Base slightly.

Inference Engine

The Inference Engine, or program driver, is that part of the system which uses the knowledge held in the Knowledge Base to come up with the results which are output from the system.

Once the basis of the overall system design had been specified, i.e. the clauses of the rules were defined in terms of each other so that comparisons could be made, an inferencing mechanism was required.

That chosen was that of a backward-chained and/or tree; a technique used by MYCIN.

As an example of how the inferencing mechanism works let us assume that we have the following rules:

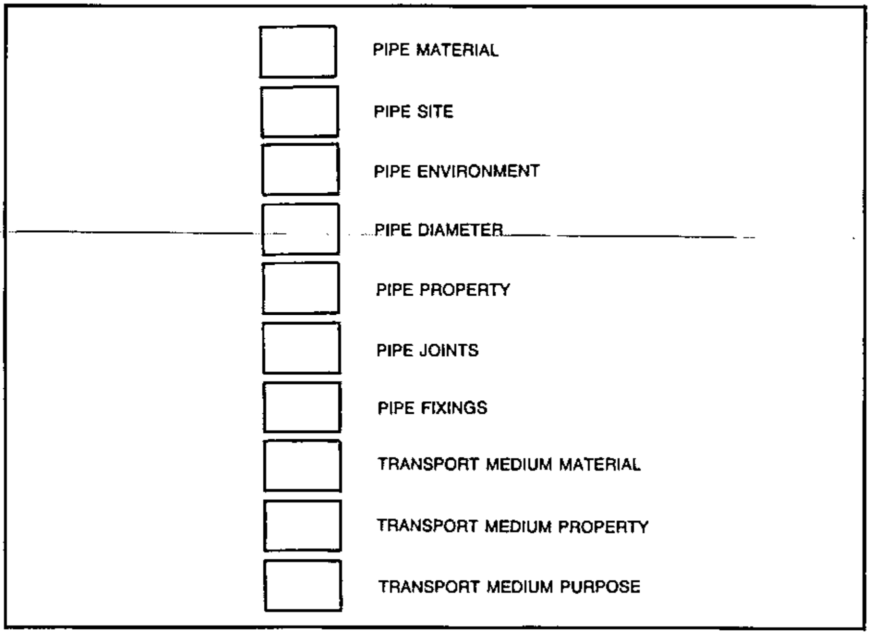
RULE A
  IF the pipe site is not buried
  AND the pipe environment is hostile
  THEN a pipe property is lagged

RULE B
  IF a transport medium property is hot
  AND the pipe environment is cold
  THEN the pipe environment is hostile

RULE C
  IF a transport medium property is cold
  AND the pipe environment is hot
  THEN the pipe environment is hostile

The system looks at Rule A. It can say that a property of the pipe is that it should be lagged *if* the pipe is not buried *and* the environment that the pipe is in is hostile. Assuming that it has already proved that the pipe is not buried; how can it show that the environment is hostile? The answer is to look at Rule B. If it can be shown that a property of the transport medium is that it is hot *and* it can be shown that the pipe environment is cold then it can be said that the environment is hostile. Once that has been done, the system can return to Rule A and deduce that a property of the pipe is that it should be lagged. Note that there may be more than one way to prove any particular point. Rule C may be of use if Rule B is not.

Diagram 2



It was stated earlier that everything PATTIE discovers is stored in the STM. Also stated was the fact that everything is defined in terms of classifications. The STM can be viewed, conceptually, as shown in Diagram 2.

System operation is easier to understand if viewed in this way. PATTIE is trying to find things to put into the 'boxes'. Anything discovered is put into the STM. Each time PATTIE tries to prove anything, the STM is looked at to see if the answer is already there. If it is not, each of the other rules in the partition being scanned (and also the 'general' partition) is looked at to find a rule that would prove the point in question. If still unsuccessful, the user is asked to provide an answer.

When all the rules in all (available) partitions have been checked then the details in the STM can be given to the user. This is achieved by scanning through the STM printing out those details.

Natural Language Interface

The interface between the user and PATTIE has been made as friendly, non-technical and easy to understand as possible. The system always has the initiative; it is working on something or asking the user a question. The user is never left alone not knowing what to type in next.

Due to the time constraint involved, it was only possible to implement a Question/Answer interface. All answers to questions are of the YES/NO variety. The, yet to be discussed, Explanation Facility required that the user was able to enter e.g., HOW 13 or WHY 45.3, at any stage in the processing. Therefore, each answer given had to be carefully vetted as to its validity.

It is our contention that the interface between the human and the computer is the most important part of the system. Apart from the obvious — if the user cannot access the system, it will be of no use — there is the equally important point that the user must see the system as a 'friend' and forget that he is talking to a machine. This is of great importance especially in the case of Expert Systems when one considers exactly what they are.

Apart from the answering of questions there is another input stage to the system. This occurs when the user is entering a new rule. As stated earlier, each rule has to be defined in terms of the classifications, thus there has to be quite strict checking of user input at this stage. The user is permitted to enter free text. However, the text is parsed and compared with the allowable classifications and the user informed exactly where in the text string an error occurred i.e., the system does more than just say 'error'.

One of the most striking features of the interface is that it follows the MYCIN example in that the system speaks in terms of YOU and I, e.g. the system says things such as "I have discovered that ....." and "Would you please enter .....".

As in the MYCIN example, the system repeats certain statements back to the user where there is potential ambiguity. The system knows what question it is answering but does the user know just what question he asked? It may sound daft but it happens.

As mentioned earlier, the concept of noise words was introduced to make the input and output of rules and associated questions as English-like as possible.

Features, such as the ability of the user to answer DON'T KNOW to a question, were considered important to implement even though the system design was complicated by this. Another feature of this type is being able to reply Y,YE,YES for YES i.e. the user can give as much of a reply as to make it unambiguous to the system.

Should the user not be sure about what a question means then he need only type HELP. Since the user is always at a question help is always available. Several levels of help were provided, where and as appropriate.

Explanation Facility

This component of the system is absolutely vital. The system *must* be able to explain itself to the user. The user must be able to ask the system questions such as "Why do you want to know that?" or "How did you reach that conclusion?".

At any stage during a session with PATTIE (either during processing or after processing has finished and the results have been returned) the user can ask the question as to WHY something was asked or HOW a conclusion was reached. All questions and conclusions output by the system are preceeded by a "statement number". When querying something, this number is used, e.g., by typing in

WHY 18

the system knows that the user wants to know why question 18 was asked. The system will respond by printing back to the user what it interprets that the user wants to know (so there will be no confusion as to the question that will then be answered), followed by the line of reasoning on the point in question.

As well as asking WHY the system wanted to know something (past tense), the user can ask WHY the system wants to know something (present tense). Similarly with asking HOW; HOW did you reach a conclusion (past); HOW will you try to reach a conclusion (future).

It is important to note that the system can, and will, always explain itself to the user.

Provision of the Explanation Facility introduced one of the most complex parts of the system. This is due to the fact that the system has been provided with the ability to look into the future to answer queries such as "How *will* you conclude that .....?".

A necessary feature of this facility is that it must be possible to obtain a complete line of reasoning, all the way back to the starting point if necessary.

Knowledge Acquisition Facility

PATTIE operates in two modes. The first is what is called "Pipe Analysis" mode. The second is a mode of operation in which it is possible to directly update or interrogate the Knowledge Base — Knowledge Base Maintenance.

Its major function is the addition of new rules to the Knowledge Base. The input format of rules has been made as 'natural' as possible within the constraints necessary to ensure an effective system. There is an editing and correction facility during the input of new rules to ensure that new rules are correctly input to PATTIE.

To allow for tasting and "discovery through playing", rules are never actually wiped from the Knowledge Base. When a rule is deleted it is marked as such but it remains in the system. The rule is ignored during any processing but it is available should it be required to restore it.

REFERENCES AND BIBLIOGRAPHY

1) Knowledge-Based Systems in Artificial Intelligence
— R. Davies & D.B. Lenat (McGraw Hill)

2) The C Programming Language
— B.W. Kerninghan & D.M. Ritchie (Prentice Hall)

3) The UNIX System
— S.R. Bourne (Addison Wesley)

4) "Expert Systems : An Introduction for the Layman"
— M.J. Winfield (Computer Bulletin (12-82))

5) "Themes and Case Studies of Knowledge Engineering"
— E.A. Feigenbaum (Expert Systems in th Microelectronic Age (1979) pp 3-25)

6) "Knowledge Engineering for Medical Decision Making : A Review of Computer-Based Clinical Decision Aids"
— Shortliffe et al (Proc. IEEE vol. 67 no. 9 (9-79)

Christopher Harvey
Ph.D. Research (Knowledge-Based Systems)
The University of Aston in Birmingham
15 Coleshill Street
Birmingham B4 7PA.

... TO BE CONTINUED IN VECTOR 4

Help Develop The Electronic Boardroom

Metapraxis is a dynamic young company producing the next wave of advanced management information systems for large corporate clients. Financed by venture capital funds and a software development grant from the DTI, we have recently completed an Operations Room in our superb new offices in South-West London equipped with the very latest in computer display technology.

We are now looking for exceptional individuals to join our development team which is using APL and operational research techniques to develop software bordering on the expert systems field. This vacancy will suit you if you like the idea of stretching yourself to your limits and working with the most advanced display hardware available.

Applicants should have an exceptional ability to pick up new ideas and be looking for a genuine intellectual and career challenge. They will either have a good honours degree (or equivalent) or have proven APL experience, preferably handling management information. They will be expected to show an impressive learning curve to extend their experience into the areas of 16-bit microcomputers, networking, telecommunications, Assembler and computer graphics.

This is a unique chance to share in and influence the growth of a new industry and to make a major contribution in an environment that is free from big company rules and regulations. Starting salary will be negotiable and should be attractive to successful applicants. Please write with comprehensive career details to David Preedy at the following address:

Metapraxis Ltd., Hanover House, Coombe Road, Kingston-upon-Thames, KT2 7AH.

METAPRAXIS

TECHNICAL SECTION

This section of VECTOR is aimed principally at those of our readers who already know APL, and will contain items to interest people with differing degrees of fluency in APL.

TECHNICAL EDITORIAL

by Dave Ziemann and Jonathan Barman

APL on micros is beset by problems of the character set. Some implementations allow the option of using keywords instead of the special APL symbols, but unfortunately each interpreter seems to have tackled the problem in a different way. For example the keywords in DEC APLSF must start with a dot, and are different from those chosen by APL*PLUS/PC which use a hash to distinguish them (although this facility can be turned off). These keywords are different yet again from those offered by MicroAPL's new implementation for Sinclair's QL computer; this uses the same keyword for both the monadic and dyadic form of a given primitive. (We hope to bring you a technical review of QL/APL in a future VECTOR.)

As if all this wasn't enough we are also driven crazy trying to remember where the special characters are on the keyboard, with major differences between the three main implementations on just one machine (APL*PLUS, Sharp-APL and IBM APL on the IBM PC). On the APL*PLUS implementation the 'composite' symbols can be generated by using the ALT key, and from version 4.0 by an overstrike sequence too. On IBM's PC APL the ALT key is also used, but many of the special characters are on different keys. Finally on Sharp's PC interpreter the composites are generated by an overstrike, non-special APL characters are to be found on the same keys as their ACSII equivalents and the standard alphabet appears on the screen in lower-case! Clearly key-cap stickers don't help if you need to make regular use of more than one interpreter.

What are the solutions to these problems? Should a standard set of keywords be adopted? Should interpreters allow user-defined symbols? Could micros with bit-mapped screens allow any symbol to be attached to any key? Let's hear your views and see if we can come up with a concensus that may help future implementations to get it right.

We wanted to make our new column 'Surely there must be a Better Way' humorous as well as useful, so we've started the ball rolling with some staggeringly bad functions and tried to show the 'correct' solutions - unless someone has any better ideas! The intention is not to poke fun at the (anonymous!) originators, but to point the way towards writing code that takes advantage of the tremendous power of APL. If you have any improvements to these, or any other, algorithms do write in and let us know. Alternatively, if you have a solution to a problem which you suspect is not optimal (or even no solution at all) please tell us and let someone else solve it for you. Additional commentary with APL code is always helpful, but if you don't have the time to write it we are willing to do this for you.

Finally, a reminder to all of you APLers out there - the technical section of VECTOR is YOUR section and should be largely driven by you. We are here to represent your views - don't let the dreaded white-space disease infect these pages! We'd like to hear from our readers and to find out who (and where) they are. so please keep those technical letters coming in. Or how about some code for 'Surely there must be a Better Way'? Or even a technical article?? Keep it coming!

Introduction to Contributed Articles

In this issue we are presenting four contributed articles.

'Generic read and replace programs for the Sharp APL file system' was written by Robert Pullman of the Rochester Group - a New York based APL consultancy. He outlines the development of a filing system that permits the retrieval and update of planes in a multi-dimensional numeric database. Although written with Sharp APL users in mind it should be easily understood by the VS APL user with a rudimentary knowledge of the Sharp enclose and disclose functions. A useful primer on the Sharp component filing system is also provided.

Following the Sharp theme, 'An APL Dialogue' is the work of Graeme Robertson from I. P. Sharp Associates, London. This is a light-hearted, yet also serious, look at hoa Socrates (the wise Greek) and Meno (the knowledgeable slave) might have arrived at Relativity Theory - with a little help from Sharp APL on a PC! Remeniscent of Hofstadter in 'Godel, Escher, Bach' Graeme nevertheless instils his own brand of humour into this impressive work. Incidentally the contents were submitted in the form of entirely plotter-produced output, but have been re-typeset for publication.

'Quotitian Jottings on Matters Temporal' is a must for those who need to store times and dates as two-byte integers, for example when using APL*PLUS/PC. We are indebted to Michael Carmicael of Occidental Oil, London, who wrote the piece, and consider ourselves extremely fortunate that we can benefit from his chance discovery in an anthropological journal.

Dr A. G. Prys Williams from the Management Science and Statistics Dept of the University College of Swansea is the author of our fourth paper 'Random Contingency Tables: a use for three-dimensional Matrices'. Non-statisticians should not be put off by the title - there are results here which can be of use to people in many areas. In particular the APL code for generating random figures with constrained row and column totals is presented and used to investigate a real problem

TECHNICAL CORRESPONDENCE

From Andrew Tarr 28th June 1984

Sir: In applications requiring a search (e.g. names in lists) APL will normally generate a boolean vector, the ones indicating locations where a match was found. Frequently the next step is to translate this into an index. FINNAPL offers three idioms to do this; one of them is the universal favourite, and while the other two present an interesting angle on the problem I cannot see any use for them as they are horribly slow by comparison:

```
FI 506:    B/⍳⍴B
FI 41:     (+/B)↑⍒B
FI 280:    (+\B)⍳⍳+/B
```

To my mind the need to convert a boolean vector to an index is common enough to make a strong case for a special primitive. This is available in DEC-10 APLSF as omega, and although it is non-standard and therefore not documented in the current manual, omega must be an early example of optimised code. Omega is defined for vectors as follows:

```
ωB ↔ B/⍳⍴B
```

It is invariably faster to execute than the equivalent compression, and because the boolean vector is its right argument, bracketing or assignment is often avoided. Unfortunately it is hard to see a sensible symbol for this primitive; as its closest relative is iota, iota-bar might be a possibility.

Perhaps surprisingly, FINNAPL only offers one idiom to count the ones in a boolean vector:

```
FI 370:    +/B
```

However (depending on the implementation) this is not necessarily the best. In APLSF, of course, the best answer is:

```
⍴ωB
```

Idiom 370 should be last choice for speed, because the DEC interpreter takes time to convert boolean to integer, so it is usually quicker to use:

```
⍴B/1
```

I attach some interesting comparisons between the idioms I have mentioned in VSAPL and in APLSF, for sparse (2.5%) and dense (50%) boolean vectors of 2000 elements.

(Times in milliseconds)	DEC-10	APLSF 2	IBM 3081K	VS APL (4.0)
	Sparse	Dense	Sparse	Dense
B/⍳⍴B	13	20	2	2.5
(+/B)↑⍒B	270	385	25	30
(+\B)⍳⍳+/B	405	7300	23	710
ωB	1	10	—	—
+/B	20	20	0.3	0.5
⍴B/1	6	12	2	2.8
+/B/1	7	20	2.3	3.0

Yours sincerely,

Andrew Tarr
ICI plc (Mond Division)
Finance and Info Systems Dept.,
The Heath,
Runcorn,
Cheshire.

From Adrian Smith 18 August 1984

Sir: You might like to add a couple of functions to the compendium of partition operations from the first issue; both these use a trailing partition, and were written to help analyse commodity data where the weekly highs and lows are important indicators.

```
      ∇ R←MSK ∆PMAX VEC
[1]    ⍝ RETURN MAXIMUM VALUES OF <VEC> WITHIN TRAILING
[2]    ⍝ PARTITIONS MARKED OFF BY <MSK>.
[3]     R←MSK/VEC[⍋VEC+(1+⌈/VEC)×-⌽+\⌽MSK]
      ∇

      ∇ R←MSK ∆PMIN VEC
[1]    ⍝ RETURN MINIMUM VALUES OF <VEC> WITHIN TRAILING
[2]    ⍝ PARTITIONS MARKED OFF BY <MSK>.
[3]     R←MSK/VEC[⍒VEC+(1+⌈/VEC)×⌽+\⌽MSK]
      ∇
```

Both the above are variations on an original theme by Elaine Gathercole.

Yours sincerely,

Adrian Smith
Operational Research
Rowntree Mackintosh plc
YORK.

From Mark Bassett 17th December 1984

Sir: The functions we were asked to find in your 'Life' competition of two issues back provide a handy way of manipulating text at the word, rather than character, level.

Partition functions are inappropriate to this kind of work due to the complexity of updating the word/non-word partition every time the text is amended. However we can use run-coding to generate a vector showing the lengths of successive words, interspersed with the lengths of the gaps between them, and this is very easy to maintain.

As an example I offer the function CHANGEWORD which performs search and replace functions on character arrays; this is superior to the functions found on some editors as it operates on words not substrings (thereby avoiding constructions such as 'dogalogue') and because it performs its substitutions in parallel it can permute words in a document without error.

One use for CHANGEWORD I have already found is a function that relabels others to bring them into line with the convention that line labels follow the sequence L1, L2, L3 ... — shades of BASIC!

```
     ∇ R←W CHANGEWORD M;A;B;C;D;I;J;L;N;O;X;Y;⎕IO
[1]   ⍝ WORD BASED SEARCH AND REPLACE FUNCTION
[2]   ⍝ Algorithm by M.S. Bassett - bugs introduced by Forces of Darkness
[3]   ⍝
[4]   ⍝ SYNTAX:
[5]   ⍝ <W> Character vector of words separated by the delimiter in W[1]
[6]   ⍝ <M> Character array to replace in
[7]   ⍝ <R> Character array like <M> with changes made as sepcified by <W>
[8]   ⍝
[9]   ⍝ DESCRIPTION:
[10]  ⍝ <W> is broken up into pairs of words; each occurence in <M> of the
[11]  ⍝ first member of a pair is replaced by the second member. If the
[12]  ⍝ second element is all spaces then the word is deleted.
[13]  ⍝ N.B. - All changes take place simultaneously, and replacement is
[14]  ⍝        performed at the word, not character, level.
[15]  ⍝      - The first element of ⎕AV must not appear in <W> or <M>
[16]  ⍝
[17]   ⎕IO←1
[18]  ⍝ <A> specifies the character set used for recognising words in <M>
[19]   A←'ABCDEFGHIJKLMNOPQRSTUVWXYZΔabcdefghijklmnopqrstuvwxyz⍙0123456789'
[20]  ⍝ Form <W> into its component words
[21]   W←W[1]FMV 1↓W
[22]   W←((0.5×1↑ρW),2,¯1↑ρW)ρW
[23]  ⍝ Identify Old and New words in <W>
[24]   O←W[;1;]
[25]   N←W[;2;]
[26]  ⍝ Add line delimiters to <M>
[27]   M←M,⎕AV[1]
[28]  ⍝ Flag word-characters in <M>
[29]   B←,M∊A
[30]  ⍝  Run-code the information in <B>
[31]   C←BTR B
[32]  ⍝ Strip out the non-words in <M> and reduce it to its component words
[33]   D←(~B)/,M
[34]   L←' ' FMV B\B/,M
[35]  ⍝ Locate old words in the list <L>
[36]   I←O XR L
[37]   J←I≤1↑ρO
[38]  ⍝ Replace old by new(this means making <L> and <N> the same width)
[39]   X←(¯1↑ρL)⌈¯1↑ρN
```

```
[40]  L←((1↑ρL),X)↑L
[41]  N←((1↑ρN),X)↑N
[42]  L[J/⍳1↑ρL;]←N[J/I;]
[43] ⍝ Now count the new word lengths in <L> and update <C>
[44]  Y←+/' '≠L
[45]  C[2×⍳ρY]←Y
[46] ⍝ Expanding <C> into its boolean form now lets us reform <M>
[47]  B←RTB C
[48]  R←B\(L≠' ')/L←,L
[49]  R←R,[0.5](~B)\D
[50]  R←(B⊖R)[2;]
[51] ⍝ Reshape <R>, preserving the line-breaks
[52]  R←⎕AV[1]FMV R
    ∇
```

```
    ∇ M←D FMV V;M;S;X;⎕IO
[1]   ⍝ FORM MATRIX FROM VECTOR
[2]   ⍝ SYNTAX:
[3]   ⍝ <D> scalar delimiter
[4]   ⍝ <V> vector delimited by <D>
[5]   ⍝ <R> Matrix whose rows are the stretches of <V> between delimiters
[6]   ⍝
[7]    ⎕IO←1
[8]    V←,V,D
[9]    S←(D=V)/⍳ρV
[10]   X←(X≠0)/X←¯1+S-¯1↓0,S
[11]   M←X∘.≥⍳⌈/0,X
[12]   M←(ρM)ρ,(,M)\(D≠V)/V
    ∇
```

```
    ∇ R←X XR Y
[1]   ⍝ MATRIX INDEXING FUNCTION
[2]   ⍝
[3]   ⍝ SYNTAX:
[4]   ⍝ <X> Array of rank ≤ 2  ( Arguments are automatically
[5]   ⍝ <Y> Array of rank ≤ 2    reshaped into matrices     )
[6]   ⍝ <R> Numeric vector giving indices of rows of <Y> in <X> (cf ⍳)
[7]   ⍝
[8]    X←(¯2↑ 1 1 ,ρX)ρX
[9]    Y←(¯2↑ 1 1 ,ρY)ρY
[10]   R←(1↓ρX)⌈1↓ρY
[11]   X←(R,1↑ρX)↑⍉X
[12]   Y←((1↑ρY),R)↑Y
[13]   R←⎕IO++/∧\Y∨.≠X
    ∇
```

M.S. Bassett
26 Falconwood Ct.
Montpelier Row
Blackheath
London
SE3 0RS.

Editor: We hope to get some feedback on Mark's function as we have not had the time to test it.

SURELY THERE MUST BE A BETTER WAY

by Dave Ziemann

We hope that this section of VECTOR will become an APL noticeboard for those small but tricky problems that one suspects have elegant solutions in APL. The trouble is that it's not always easy to find these solutions — so if you have a problem like this, send it in to us for publication: maybe someone else can solve it for you! Alternatively, if you see a familiar problem in these pages, let's see your solution to it — you never know, it might be the best one. In this way we hope to provide a source of APL idioms and algorithms for the APL community. If you do send us some code, please remember to abide by the VECTOR publication standards and try to use that funny little lamp symbol, especially in the first few lines of the functions.

For a first attempt, we will take a rather light-hearted look at what we believe to be two extreme cases of "Surely There Must be a Better Way". These actual examples were extracted from live applications systems, and indicate possible ways in which APL can get a bad name among the programming languages.

The first is the following uncommented function which was taken from a financial workspace:

```
    ∇ R←CURRDEF;X
       X←CURRENCY
       →(X[1]=0)/COMB
       →(X[1]=1)/STER
       →(X[1]=2)/DOL
      COMB:→(X[2]=1)/CSTER
       →(X[2]=2)/CDOL
      CSTER:R←5
       →0
      CDOL:R←6
       →0
      STER:→(X[2]=1)/SSTER
       →(X[2]=2)/SDOL
      SSTER:R←1
       →0
      SDOL:R←2
       →0
      DOL:→(X[2]=1)/DSTER
       →(X[2]=2)/DDOL
      DSTER:R←5
       →0
      DDOL:R←6
       →0
    ∇
```

By constructing a table of all the possible input/output combinations

```
    | 1  2
 ---|------
  0 | 5  6
  1 | 1  2
  2 | 5  6
```

it doesn't take too long to come up with this original solution, for an input VECTOR C:

```
R← 5 6 1 2 5 6 [2⊥C]
```

For those who prefer a more arithmetic approach,

```
R←C[2]+4×C[1]≠1
```

will do the trick. If you like an origin-independent solution, then the following is for you:

```
R←1↓C+⌽4×C≠1
```

In generating these alternatives, we should not forget the value of using APL comment lines to explain WHAT it is we are trying to do. For lines of code that are non-obvious, a further comment explaining how this is being achieved is also a good idea.

Our second example is this remarkable function for replacing non-underscored alphabetics by their equivalent underscored characters:

```
      ∇ R←UNDERSCORE1 VECTOR
[1]     VECTOR[POSITION VECTOR='A']←'A̲'
[2]     VECTOR[POSITION VECTOR='B']←'B̲'
[3]     VECTOR[POSITION VECTOR='C']←'C̲'
[4]     VECTOR[POSITION VECTOR='D']←'D̲'
[5]     VECTOR[POSITION VECTOR='E']←'E̲'
[6]     VECTOR[POSITION VECTOR='F']←'F̲'
[7]     VECTOR[POSITION VECTOR='G']←'G̲'
[8]     VECTOR[POSITION VECTOR='H']←'H̲'
[9]     VECTOR[POSITION VECTOR='I']←'I̲'
[10]    VECTOR[POSITION VECTOR='J']←'J̲'
[11]    VECTOR[POSITION VECTOR='K']←'K̲'
[12]    VECTOR[POSITION VECTOR='L']←'L̲'
[13]    VECTOR[POSITION VECTOR='M']←'M̲'
[14]    VECTOR[POSITION VECTOR='N']←'N̲'
[15]    VECTOR[POSITION VECTOR='O']←'O̲'
[16]    VECTOR[POSITION VECTOR='P']←'P̲'
[17]    VECTOR[POSITION VECTOR='Q']←'Q̲'
[18]    VECTOR[POSITION VECTOR='R']←'R̲'
[19]    VECTOR[POSITION VECTOR='S']←'S̲'
[20]    VECTOR[POSITION VECTOR='T']←'T̲'
[21]    VECTOR[POSITION VECTOR='U']←'U̲'
[22]    VECTOR[POSITION VECTOR='V']←'V̲'
[23]    VECTOR[POSITION VECTOR='W']←'W̲'
[24]    VECTOR[POSITION VECTOR='X']←'X̲'
[25]    VECTOR[POSITION VECTOR='Y']←'Y̲'
[26]    VECTOR[POSITION VECTOR='Z']←'Z̲'
[27]    R←VECTOR
      ∇

      ∇ R←POSITION B
[1]     R←B/⍳⍴B
      ∇
```

Of course, the same effect could be achieved by looping through each letter of the alphabet, but the absence of a loop in this function may be an indication that the author once heard that looping in APL was inefficient! Because the function only works on vectors, function fragments of the following kind are found elsewhere in the workspace:

```
                 .
                 .
                 .
                [24]   LARGEMAT[1;]←UNDERSCORE1,LARGEMAT[1;]
                [25]   LARGEMAT[3;]←UNDERSCORE1,LARGEMAT[3;]
                [26]   LARGEMAT[6;]←UNDERSCORE1,LARGEMAT[6;]
                [27]   LARGEMAT[11;]←UNDERSCORE1,LARGEMAT[11;]
                [28]   LARGEMAT[14;]←UNDERSCORE1,LARGEMAT[14;]
                [29]   LARGEMAT[15;]←UNDERSCORE1,LARGEMAT[15;]
                 .
                 .
                 .
```

For those of you new to APL a non-looping rank-independent solution can be formulated as follows:

```
    ∇ R←UNDERSCORE2 W;S;I;J
[1]   ⍝ <R> IS CHARACTER ARRAY <A> WITH ALPHABETICS REPLACED BY UNDERSCORES.
[2]   ⍝ METHOD: LOOK UP ALL CHARACTERS AND THEN ONLY REPLACE ALPHABETICS.
[3]    S←⍴W
[4]    R←,W
[5]    I←'ABCDEFGHIJKLMNOPQRSTUVWXYZ'⍳R
[6]    J←(I≠26+⎕IO)/⍳⍴I
[7]    R[J]←'ABCDEFGHIJKLMNOPQRSTUVWXYZ'[I[J]]
[8]    R←S⍴R
    ∇
```

The function can be applied to any rank array, and so can be used on sets of VECTORS in the following way:

```
                        .
                        .
                        I←1 3 6 11 14 15
                        LARGEMAT[I;]←UNDERSCORE LARGEMAT[I;]
                        .
                        .
```

An alternative function which first restricts the set of characters to the alphabetics, and then maps them to underscored letters can also be written:

```
    ∇ R←UNDERSCORE3 W;S;I;J;A
[1]   ⍝ <R> IS CHARACTER ARRAY <A> WITH ALPHABETICS REPLACED BY UNDERSCORES.
[2]   ⍝ METHOD: RESTRICT SET TO ALPHABETICS, THEN REPLACE BY UNDERSCORES.
[3]    S←⍴W
[4]    R←,W
[5]    I←(R∊A←'ABCDEFGHIJKLMNOPQRSTUVWXYZ')/⍳⍴R
[6]    J←A⍳R[I]
[7]    R[I]←'ABCDEFGHIJKLMNOPQRSTUVWXYZ'[J]
[8]    R←S⍴R
    ∇
```

In many cases this function will run faster than the one above, depending on the interpreter you are using.

These functions were not included as objects of ridicule, but as a reminder for all of us to be very careful when writing APL — given a powerful tool we must demonstrate our control over it, not by abuse, but by careful application and, occasionally, restraint. Hopefully these examples will inspire us all to improve the quality of our APL code.

PRIZE COMPETITION RESULT: This is your Life

by David Ziemann

The first VECTOR competition attracted respondents from Austria, Holland and Switzerland, as well as from the UK. The UK entries even included one from a certain P Andrew with an address in The Mall, London!

To recap, the problem was to write two monadic functions BTR and RTB that convert between the boolean and run-coded representations of a binary grid. For example:

```
          BM
 0  0  1  1  0
 0  0  0  1  1
 1  1  1  0  0
 1  1  0  0  0

          BTR  ,BM
2  2  4  5  2  2  3

          (ρ,BM)∧.=+/BTR  ,BM
1

          (,BM)∧.=RTB  BTR  ,BM
1
```

The first element of the run-coded vector was to always give the number of leading zeros in the corresponding ravelled boolean matrix. The two functions were to be written in ISO Standard APL and were also to be independent of their environment. All else succeeding, entries were to be judged on the brevity of the solutions.

There were many interesting responses, including one recursive function and one entry with six rather than two functions!

So, what happened? Well, each entrant's functions were automatically tested with sixteen different arguments to see how they behaved. These cases included boolean matrices of all types — leading ones, leading zeros, trailing ones, trailing zeros, singular matrices, empty matrices, matrices of different sizes, etc., etc. In each case both functions were tested with an external index origin setting of both zero and then one, to check for environment independence.

The result of all this was that some functions were disqualified for producing incorrect results. For example: three RTBs, and two BTRs when the global index origin was zero, one BTR for leading zeros in the argument and four BTRs for empty matrices. One BTR was disqualified for producing an APL error with an empty argument.

Note that an empty argument to BTR should produce a vector whose only element is a zero, in order for the above identities to hold.

About half of the BTRs achieved origin independence by explicitly setting quadIO first. A more concise answer is possible if the system variable is referenced explicitly, as in this 31 character solution by Thomas van den Heuvel:

```
      ∇ R←BTR B;X;Y
[1]    ⍝ CONVERT BOOLEAN VECTOR <B> TO RUN-CODE VECTOR <R>
[2]     R←Y-¯1↓⎕IO,Y←X/⍳⍴X←(B≠¯1↓0,B),1
      ∇
```

Further research shows that a shorter solution is possible if one remembers that the function argument is always boolean:

```
      ∇ R←BTR B
[1]    ⍝ CONVERT BOOLEAN VECTOR <B> TO RUN-CODE VECTOR <R>
[2]     R←R-⎕IO,¯1↓R←R/ιρR←(0,B)≠B,2
      ∇
```

Mike Day had the gall to suggest this incredible 25 character looping solution:

```
      ∇ R←BTR B;C
[1]    ⍝ CONVERT BOOLEAN VECTOR <B> TO RUN-CODE VECTOR <R>
[2]     R←''
[3]     R←R,+/~C←∨\B
[4]     →3⌈B←~C/B
      ∇
```

The judges do not recommend the use of this particular function in a real application!

Andrew Tarr pointed out that users of DEC-10 APLSF could increase BTR's efficiency, and further shorten it, by using monadic omega. This function directly converts a boolean vector into an index vector, as follows:

```
ωB ←→ B/ιρB
```

(See the technical letters section for Andrew's letter — Ed.)

The RTBs were generally more interesting. A number of entrants used indexing to solve the problem, whereas others used outer products of various kinds (less than, less than or equals, greater than or equals, plus) followed by reductions or compressions to produce the desired result. Those who strove for a shorter solution discovered that outer product was not necessary. At 18 characters, the shortest and most elegant RTB was submitted by Phil Last:

```
      ∇ R←RTB V
[1]    ⍝ CONVERT RUN-CODE VECTOR <V> TO BOOLEAN VECTOR <R>
[2]     R←≠\(ι+/V)∈⎕IO++\V
      ∇
```

A surprising number of entrants used the residue function, failing to recall the following boolean identities:

```
2|+/B ←→ ≠/B
2|+\B ←→ ≠\B
```

Claude Henriod and Phil Last pointed out that RTB is trivial with an APL that supports extended compression (replicate):

```
      ∇ R←RTB V
[1]     R←V/(ρV)ρ 0 1
      ∇
```

No doubt some authors arrived at their solutions by working backwards from this one.

After deliberation the judges decided to award the prize money to Phil Last (£20), Mike Day (£20) and Thomas van den Heuvel (£10). Special commendations also go to Mark Bassett and J Jollife.

Two entrants wanted to know how the run-coded representation was used to calculate the next generation in the "Game of Life". Space prohibits the full details of the algorithm, which was due to Paul Chapman, being printed here.

Briefly though, the rows of the boolean matrix are individually scanned from left to right with concurrent reference to the run-code vector. In this way subsequent columns of the next generation grid are generated. This finite-state machine approach is obviously heavily reliant on loops, but APL was only used as a prototype for the final machine code version. One of the main advantages of this algorithm is that it can deal with large amounts of empty space in one step.

E&S ASSOCIATES

THE APL PROTOTYPING EXPERTS

• CONSULTANCY • SOFTWARE •

PROTOPAK

PROVIDES PROTOTYPING FACILITIES IN:

* Relational Databases
* Multi-dimensional arrays
* Financial modelling
* Full Screen editing and formating (AP 124)

TRANSPARENT APL - SIMPLE QUERY LANGUAGE FOR IBM PC AND MAINFRAME USERS.

E&S Associates
19 Homesdale Road
Orpington Kent
BR5 1JS

0689 24741

PRIZE COMPETITION: Test your skill

by Dave Ziemann

A company specialising in APL consultancy has a number of staff, each of whom has talents in a small number of different areas. For example, John has experience in system design, the TSO and CMS operating systems, and in APL*plus/PC. Anne on the other hand, is skilled in all of these and also in APL2 and SHARP APL. Bill has APL*plus/PC and APL2 whereas Mary has system design, SHARP APL, mathematical techniques, CMS and APL2.

This situation is represented by a consultants' skills matrix as follows:

```
        ←CONS←4 6ρ1 2 3 7 0 0, 1 3 7 9 2 6, 7 9 0 0 0 0, 1 6 5 3 9 0
1 2 3 7 0 0
1 3 7 9 2 6
7 9 0 0 0 0
1 6 5 3 9 0
```

where each row represents one consultant's skills, and each non-zero entry is an index into a table of skill descriptions. Notice that skill 3 (CMS) belongs to John, Anne and Mary, and that in this representation zeros are used to pad the shorter skills vectors.

Now, when some work comes up, the skills of each consultant are matched against the skills required by the customer. Naturally the consultant must have all those talents needed for the job. Because business is good many jobs come in at once, and this is expressed as a jobs skills matrix each row of which contains the skill numbers required for the job; for example:

```
        ←JOBS←5 3ρ1 2 3, 7 9 0, 1 4 0, 1 3 0, 1 6 3
1 2 3
7 9 0
1 4 0
1 3 0
1 6 3
```

In the attempt to allocate consultants to jobs, the company wants to generate a boolean matrix which will indicate the jobs that can be tackled by each consultant.

The competition is to write a dyadic function called <SKILLSMATCH> which will produce the boolean matrix as output. The left argument must be the jobs matrix and the right argument the consultants matrix. The result must then be a boolean matrix with one row per job and one column per consultant, where a 1 indicates that the consultant has sufficient skills to do the job. For example:

```
        JOBS SKILLSMATCH CONS
1 1 0 0
0 1 1 0
0 0 0 0
1 1 0 1
0 1 0 1
```

The company in question has a large number of customers and employees, whereas the number of unique skills is low. Entrants will therefore not be penalised for offering solutions that assume no more than 10 skills, but the demands on workspace, speed of execution, robustness and degree of intelligibility will be the major factors in deciding the winners.

To enter the competition you must write ISO standard-conforming code (i.e. no special features like nested arrays or new quad-functions), but we are always interested to see (and publish) alternative solutions written in other APLs.

The closing date for entries is 30th June 1985.

Competition Rules

— Entries must be in legible English or APL as appropriate and should preferably be machine produced.

— Entrants must declare the type of computer and the version and release level of the APL interpreter on which their functions were written.

— The date and your full name and address should appear on each sheet of your entry.

— Entries should be physically separate from other contributions such as letters, and should be clearly marked 'Competition Entry'.

— All submissions should be sent to the editor.

— Those on the committee, activities group or journal group of the British APL Association are ineligible.

GENERIC READ AND REPLACE PROGRAMS FOR THE SHARP APL FILE SYSTEM

by Robert Pullman

Introduction

This paper presents a simple design for numeric APL data bases. The design is quite general. Filing and retrieving from any such data base is handled by a very small module. The design is practical for applications which deal with several data bases.

Unfortunately, one cannot write about the APL file system but rather an APL file system. Files, or lack of, were an early divergence node for APL implementations. Each vendor has done it differently, and IBM has not done it at all. Allowing that a fair number of readers are only familiar with the IBM flavours of APL, we begin with a brief exposition of APL files. Since there is no standard, I'll present the one that I'm most familiar with, the SHARP APL file system.

A primer on (SHARP) APL files

The great convenience of APL files is that one can work with them directly (no auxiliary processor) and programmatically (without system commands). The problem of data storage and shared access is thus brought within the APL environment. All the tools for working with the file system are present in every workspace as system functions.

While not all applications require files, nearly all business applications do. If several different workspaces deal with the same data base, there is a logistical problem of collecting (filing) and distributing (reading). One solution is to use)COPY and)SAVE, which is quite awkward to implement smoothly even if all users are from one account. IBM's solution is to have non-APL programs perform the file I/O, and to employ shared variables and auxiliary processors to interface with APL. This isn't at all a 'bad' solution, but it takes one out of the APL environment and requires expertise in the non-APL world.

The usefulness of APL files as a third alternative was apparent enough to vendors such as STSC, SHARP, XEROX, and DEC, all of whom had APL file systems in the early 70's. Today nearly all except IBM have file systems. Some file systems closely resemble others, as SHARP's resembles STSC's, but no two are entirely alike.

The simplest way to describe an APL file is *a place to store variables.* Any variable in any workspace can be put anywhere on any file. File records are called *file components.* A file is a string of components. Each component has a *file index* (also called *component number*) which is used for direct access to the component.

For example, a file of sales data for 80 sales locations might be organized:

File Index	Location
1	LONDON
2	NEW YORK
3	PARIS
4	MADRID
.	.
.	.
80	BRUSSELS

And each component of the file might be a matrix of unit sales figures for 54 products and 12 months.

Each APL account has a *file library*. Suppose that the sales data is in account 123's library under the name 'SALES'. To use the data base, one would *share tie* (open) the file and *read* (retrieve) from it. If we wanted the New York data for all products for the first three months:

```
'123 SALES' ⎕STIE 1
←(⎕READ 1 2)[;1 2 3]
```

Several files may be share tied at the same time. The right argument to quadSTIE established a distinct *file tie number*. The tie number is used to qualify which file to quadREAD from. The argument to quadREAD is tie number (1), file index (2). Most of the systems functions that pertain to files take tie numbers as one or part of one argument.

If there's a new sales location to be added to the file from a variable, X, in the workspace,

```
X ⎕APPEND 1
```

would append it to the end of the file that's tied to 1. If we have updated sales data for London in variable Y,

```
Y ⎕REPLACE 1 1
```

Would do the job. When we're all done with '123 SALES', we close it by

```
⎕UNTIE 1
```

There are some subtleties. If 1 was already in use as a tie number for a different file, a different tie number would have to be chosen. If '123 SALES' was already open, we'd need to know by which tie number. These things matter but are not important enough to dwell on.
There are quite a few system functions involved, quadCREATE, quadSIZE, quadRDAC,quadERASE,quadDROP..... The one that we're most concerned with is quadREAD for the moment.

The Design

'123 SALES' has 80 components, each 54 x 12. The file can be thought of as an 80 x 54 x 12 array, partitioned on the first dimension. That is the essence of the design. The problem is to relate a physical file structure to a logical view of it as a simple array.

If we think of the file as an 80 x 54 x 12 array, London would be the first plane, New York the second, and so forth. If we wanted to subscript the logical array, A, as follows:

```
A[2 4;5 6 7;1 2 3]
```

we would take

```
(⎕READ 1 2)[5 6 7;1 2 3]
```

```
(⎕READ 1 4)[5 6 7;1 2 3]
```

We could write the following program to handle any combination of components (planes), rows, and columns:

```
[1] ⎕UNTIE ⎕NUMS ⍝ CLOSE ALL FILES
[2] '123 SALES' ⎕STIE 1
[3] DATA←((⍴COMPONENTS),(⍴ROWS),⍴COLUMNS)⍴0
[4] I←1
[5] LOOP:DATA[I;;]←(⎕READ 1,COMPONENTS[I])[ROWS;COLUMNS]
[6] I←I+1
[7] →LOOP IF I≤⍴COMPONENTS
[8] ⎕UNTIE 1
```

Not too difficult. We can make 'DATA' the explicit result of the function and 'I' a local variable. We're left with 'COMPONENTS' 'ROWS', and 'COLUMNS' to pass to the function as an argument. The most direct way is to pass them as a three element enclosed vector and unpack them within the function:

```
[.3] COMPONENTS←>ARGUMENT[1]
[.4] ROWS←>ARGUMENT[2]
[.5] COLUMNS←>ARGUMENT[3]
```

We might also make the file name the left argument of the function and rewrite [2] with the file name as the left argument of quadSTIE. We would then have a read function that would work on other files with the same structure as '123 SALES'.

While we have a more general program, we're still restricted to matrix components and a simple file layout. Suppose, for example, that there were 6 years of data on '123 SALES'. The conceptual array would be 6 x 80 x 54 x 12, or some permutation thereof. We'd either have three-dimensional components or a two-dimensional file layout. Either way the program would no longer work. Our goal is to modify the program so that it will work regardless of the rank of the components or the rank of the file layout. In the next section we proceed toward this goal in two steps. We first modify the program to work regardless of the file layout, then to work regardless of component rank.

Generalization

Suppose again that there are 6 years of data on '123 SALES', in sets of 80 components:

			LOCATION			
YEAR	LONDON	NEW YORK	MADRID	PARIS	...	BRUSSELS
1979	1	2	3	4		80
1980	81	82	83	84		160
.						
.						
1984	401	402	403	404		408

This table shows the file index for each combination of year and location. If we were after 1981, 1982 for Paris, London we would read from components 164,161 (1981) and 244,241 (1982).

In the logical array, we are subscripting by rows 3, 4 and columns 4, 1.

Translation from logical subscripts to file indices is an application of base value. For example:

```
              1+6 80⊥2 3
        164
              1+6 80⊥3 0
        241
```

More generally,

```
FILEINDEX←1+FILELAYOUT⊥SUBSCRIPT-1
```

If the file layout is 6 80 and we want to find the file index for 1983, Madrid, we plug 5 3 into SUBSCRIPT and 6 80 into FILELAYOUT to get 323.

Let's look at a revised program that would handle the hashing of file indices from logical subscripts:

```
[1]  SUBSCRIPTS←>ARGUMENT[1]
[2]  LAYOUT←>ARGUMENT[2]
[3]  SHAPE←SHAPEOF SUBSCRIPTS
[4]  COMPONENTS←LAYOUT HASH(ρLAYOUT)↑SUBSCRIPTS
[5]  SUBSCRIPTS←¯2↑SUBSCRIPTS
[6]  ROWS←>SUBSCRIPTS[1]
[7]  COLUMNS←>SUBSCRIPTS[2]
[8]  ⎕UNTIE ⎕NUMS
[9]  FILENAME ⎕STIE 1
[10] I←1
[11] DATA←((ρCOMPONENTS),(ρROWS),ρCOLUMNS)ρ0
[12] LOOP:DATA[I;;]←(⎕READ 1,COMPONENTS[I])[ROWS;COLUMNS]
[13] I←I+1
[14] →LOOP IF I≤ρCOMPONENTS
[15] DATA←SHAPEρDATA
```

We're still assuming that all files have two-dimensional components, and that the filename is given as a left argument. There are 3 new twists. There is a subroutine, SHAPEOF, that tells us what the ultimate shape of the data ([15]) should be. 'SUBSCRIPTS' is an enclosed vector of subscripts for the logical array. SHAPEOF just returns the shape of each element of the enclosed vector.

On line [3] we store the shape, on line [15] we reshape. All that hash does is to calculate a vector of file indices for cells of the logical array. The base value algorithm is used as above. The remaining trick is to produce the right argument to base value, which is always a matrix with as many rows as elements in LAYOUT.

Each column of the matrix is a combination of indices for each file layout dimension.

For example, if the file layout is 6 × 80, years × locations, and we want years 2 and 3, locations 3,4 and 5, the matrix would have 2 rows (row 1 = years, row 2 = locations) and 6 columns. The columns would be, respectively, 2 3,2 4,2 5,3 3,3 4, and 3 5. Not coincidentally, the columns preserve row major order. The result of hash would be a six-element vector of file indices — 83 84 85 163 164 165.

The number of rows of the matrix is always equal to the shape of LAYOUT, and each row corresponds to a dimension of the file layout (years and locations in our example). The number of columns in the matrix is always equal to

```
×/(ρLAYOUT)↑SHAPE
```

as defined in the program. Here is one way of writing the hash program.

```
      ∇ COMPONENTS←LAYOUT HASH SUBSCRIPTS
[1]     SUBSCRIPTS←CROSSCATENATE SUBSCRIPTS
[2]     COMPONENTS←1+LAYOUT⊥SUBSCRIPTS-1
      ∇


      ∇ NTUPLES←CROSSCATENATE VECTOR;SHAPE;RESHAPE;REPLICAS;INDEX;⎕IO
[1]     ⎕IO←1
[2]     SHAPE←SHAPEOF VECTOR
[3]     RESHAPE←×/SHAPE
[4]     NTUPLES←((⍴VECTOR),RESHAPE)⍴0
[5]     REPLICAS←⌽1,×\⌽1↓SHAPE
[6]     INDEX←1
[7]    LOOP:NTUPLES[INDEX;]←RESHAPE⍴REPLICAS[INDEX]/>VECTOR[INDEX]
[8]     INDEX←INDEX+1
[9]     →LOOP IF INDEX≤⍴VECTOR
      ∇
```

Perhaps I should have made some mention that the examples assumed quadIO of 1. In the Appendix is a full listing of the final program in origin independent form.

We now have a program that can work regardless of file layout, but is restricted to two-dimensional components. The generalization to components of arbitrary rank could be done in much the same way as file layout was, e.g.

```
[   ] LOOP:∆←⎕READ 1,COMPONENTS[I]
[   ] ∆←,∆
[   ] DATA[I;]←∆[SUBSCRIPTS]
```

With 'SUBSCRIPTS' derived appropriately. The program can now handle any rank file layout, and rank component.

The key principles are row major order and properties of reshape. The process is certainly easier to write with enclosed arrays. Usage of enclosed arrays needn't be more than enclosing and disclosing to pass and unpack arguments. There are quite a few ways to write the program, depending on how much one wants to use enclosed arrays. The full listing here uses them minimally for the sake of simplicity. Any motivated APLer should be able to find bigger and better ways to do the same thing.

Practicality and extensions

There are two constraints posed by this design: all components must be identically shaped, and the program retrieves monolithically. If the logical array is sparse, the storage cost might not be worth it. If the idea is to return a single number as a function of each component, the program might pose workspace full problems by returning subsets of each component instead of single numbers.

The sparseness problem is rather black and white. Either it is or it isn't. The latter problem can be dealt with by modifying the program. The code does not take up very much space. There could be several different versions of the program to handle different situations.

In practice I have found that the fundamental idea is robust enough that it can be extended in many different directions, some of which are:

1) Multiple-file data bases.
2) A 'data origin' parameter, as in

```
COMPONENTS←ORIGIN+LAYOUT⊥SUBSCRIPTS-⎕IO
```

   This allows one to use the first (origin-1) component for things like dictionaries.
3) Filing of data, which is nothing more than a transposition of the left and right of the assignment.
4) Creating data bases.
5) Integrating file operations with macros.

It is not very hard to integrate the file I/O module with equally generic modules for dictionary maintenance and other data base chores. I have sort of followed my own nose in building a multiple-data base management system upon a file I/O module. The system is conversational and oriented somewhere between an application generator for programmers and a multiple data base spreadsheet (of sorts) for end users. To my surprise it has become popular with both, primarily because it reduces file I/O to a handful of simple parameters.

Appendix — Function Listings

```
    ∇ DATA←TIE RETRIEVE ARGUMENT;SUBSCRIPTS;LAYOUT;SPLIT;ORIGIN;SHAPE;COMPON
ENTS;STOP;INDEX
[1]   UNPACK
[2]  ⍝ INSTEAD OF PASSING THE FILE NAME, THE FILE IS ALREADY TIED
[3]  ⍝ AND THE FILE TIE NUMBER IS PASSED AS THE LEFT ARGUMENT
[4]  ⍝
[5]   SHAPE←SHAPEOF SUBSCRIPTS
[6]  ⍝
[7]   HASH
[8]  ⍝
[9]   STOP←⎕IO+⍴COMPONENTS
[10]  DATA←((⍴COMPONENTS),(⍴SUBSCRIPTS))⍴0
[11]  INDEX←⎕IO
[12] LOOP:
[13]  DATA[INDEX;]←(,⎕READ TIE,COMPONENTS[INDEX])[SUBSCRIPTS]
[14]  INDEX←INDEX+1
[15]  →LOOP IF INDEX<STOP
[16]  DATA←SHAPE⍴DATA
    ∇
```

```
    ∇ UNPACK;⎕IO
[1]   ⎕IO←1
[2]   SUBSCRIPTS←⊃ARGUMENT[1] ⍝ AS BEFORE
[3]   LAYOUT←⊃ARGUMENT[2] ⍝ EXTENDED TO INCLUDE COMPONENT SHAPE
[4]   SPLIT←⊃ARGUMENT[3] ⍝ THE RANK OF THE FILE LAYOUT
[5]   ORIGIN←⊃ARGUMENT[4] ⍝ FILE INDEX ORIGIN FOR DATA
[6]  ⍝
[7]  ⍝ IN TERMS OF THE EXAMPLE, <LAYOUT> WOULD BE 6 80 54 12.
[8]  ⍝ <SPLIT> WOULD BE 2 (FILE LAYOUT OF 6 80).
[9]  ⍝ AND <ORIGIN> WOULD BE 1 (FIRST FILE INDEX FOR DATA).
    ∇
    ∇ HASH
[1]   ⍝
[2]    COMPONENTS←ORIGIN+(SPLIT↑LAYOUT)⍳(CROSSCATENATE SPLIT↑SUBSCRIPTS)-⎕IO
[3]   ⍝
[4]    SUBSCRIPTS←⎕IO+(SPLIT↓LAYOUT)⍳(CROSSCATENATE SPLIT↓SUBSCRIPTS)-⎕IO
[5]   ⍝
    ∇


    ∇ TUPLES←CROSSCATENATE VECTOR;SHAPE;RESHAPE;REPLICAS;INDEX;⎕IO
[1]   ⍝ JUST SOME SIMPLE REPLICATION
[2]    ⎕IO←1
[3]    SHAPE←SHAPEOF, ¨ >VECTOR
[4]    RESHAPE←×/SHAPE
[5]    TUPLES←((⍴VECTOR),RESHAPE)⍴0
[6]    REPLICAS←⌽1,×\⌽1↓SHAPE
[7]    INDEX←1
[8]   LOOP:TUPLES[INDEX;]←RESHAPE⍴REPLICAS[INDEX]/>VECTOR[INDEX]
[9]    INDEX←INDEX+1
[10]   →(INDEX≤⍴VECTOR)/LOOP
    ∇


    ∇ SHAPE←SHAPEOF VECTOR
[1]   ⍝ RETURNS THE LENGTH OF EACH ELEMENT OF A VECTOR OF VECTORS
[2]    SHAPE←,>1↑ ¨ >⍴ ¨ >VECTOR
[3]    SHAPE←(SHAPE≠0)/SHAPE ⍝ COMPRESS OUT DEGENERATE CASES
    ∇


    ∇ R←L IF C
[1]    R←C/L
    ∇
```

AN APL DIALOGUE
(between Socrates, the wise Greek, and Meno, the knowledgeable slave)

by G. D. Robertson

Socrates: Salutations Meno! What brings you here today?

Meno: O Socrates [1], hello master. Can you pacify my troubled mind please. I have been talking to Protagoras about Rationalized APL[2] and neither of us can see how this notation could help us to understand why the gods have placed us here.

S: Does Greek help you in this respect Meno?

M: We agree that it does sir.

S: Then is it not proper that A Programming Language, sharpened by mathematical thinking, and resting firmly upon precision in interpersonal communication, likewise should contribute to that noble goal: and that in no small or shallow degree?

M: It is proper sir: but how?

S: If APL is to help in the explication of nature, we must begin with physical facts. Tell me an interesting fact Meno.

M: I have heard a very strange report. It isn't an article of faith required by the temple and the oracle at Delphi hasn't said it either.

S: What is it Meno?

M: Well, Achilles [3], as you know, can run very very fast. He claims that if he measures how fast a light beam from a fire moves by him when he is at rest, and then again measures another beam from that fire when he is running at full speed towards it, he gets exactly the same result, C, where

```
      C←871126392 ⍝ MILES PER HOUR
```

He says that the speed of light is always the same, no matter how fast he runs either towards the light or away from it. His tortoise friend also gets the same result when Achilles moves at different speeds holding a candle. In one hour a light beam always travels a length equal to 871,126,392 miles in any frame of reference at rest or uniformly moving with respect to the Earth.

```
      'FACT'⊢C=871126392 ⍝ C IS THE RESULT OF A MEASUREMENT
```

S: Crikey! That is indeed remarkable Meno.
We should more readily give credence to immediate experience of nature and its natural interpretation than to the most cogent a priori deductions. Therefore, trusting Achilles, let us accept this observation as fact.

M: If you say so master.

S: Not because I say so Meno, but because the fact has been verified by Achilles' experience and we can't run fast enough to corroborate it.
How do you define length Meno?

M: Well: Pythagoras [4] said that the length of the hypotenuse of a right angled triangle squared is equal to the sum of the squares of the other two sides.

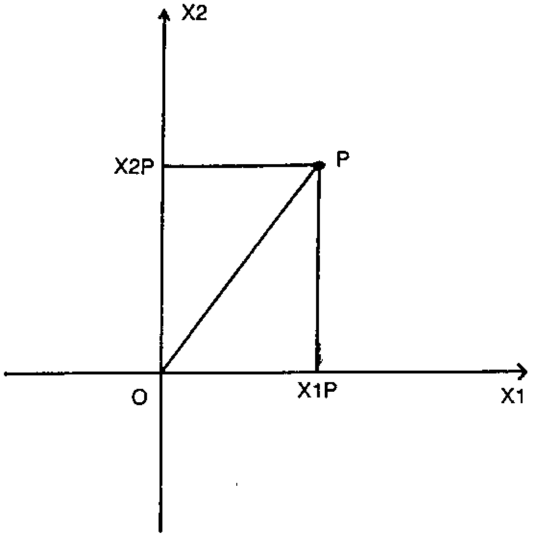
S: Express that in APL please and draw a diagram.

M: OK. In two dimensions, a Democritean [5] atom at point P has a position described exactly in APL by the two element vector

```
P←X1P,X2P
```

and the length of the line from the origin to P is

```
(+/P*2)*÷2
```



S: I see. So for any atom in this two dimensional space, given your stationary system of reference, you can define a function which determines the length of the line joining the origin of the reference frame and the point occupied by the atom.

M: Yes. Using the new symbol

```
⧐
```

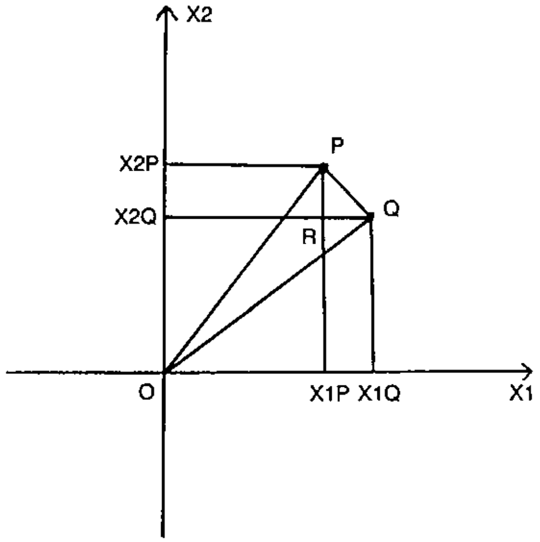
which I call Line.

```
⧐←'(+/⍵*2)*÷2' ∇ ''
```

S: What about two atoms, one at point P and the other at point Q? What is the distance between them?

M: OK.

```
P←X1P,X2P
Q←X1Q,X2Q
```



The distance PQ is equal to the square root of RQ squared plus RP squared. The distance RQ is the difference between the X1 coordinates of P and Q while RP is the difference in their X2 coordinates. So the distance between P and Q is

```
(+/(P-Q)*2)*÷2
```

We can generalize the function Line so that monadically it gives the length of the line joining a point to the origin of the reference frame, while dyadically it gives the distance between two points.

```
⊫←'(+/⍵*2)*÷2' ∇ '(+/(⍺-⍵)*2)*÷2'
```

If you don't like parentheses the ambivalent function can be written

```
⊫←'.5*⊂+/⍵*2' ∇ '.5*⊂2*⊂+/⍺-⍵'
```

S: That's very interesting Meno. Your function applies to three dimensions as well. And if you have lots of atoms, the positions of each being described in a row of a matrix, your function will give meaningful results here too.

M: Oh!

S: The laws of vector algebra are satisfied in APL. Given scalars M and N, and vectors P, Q and R.

Addition of vectors is commutative.

```
'TRUE'⊢(P+Q)≡(Q+P)
```

Addition of vectors is associative.

```
'TRUE'⊢(P+(Q+R))≡((P+Q)+R)
```

Multiplication of a vector by a scalar is commutative.

```
'TRUE'⊢(M×P)≡(P×M)
```

Multiplication of a vector by scalars is associative.

```
'TRUE'⊢(M×(N×P))≡((M×N)×P)
```

Multiplication of a vector by scalars distributes over addition.

```
'TRUE'⊢((M+N)×P)≡((M×P)+(N×P))
```

Multiplication of a scalar by vectors also distributes over addition.

```
'TRUE'⊢(M×(P+Q))≡((M×P)+(M×Q))
```

Can you write functions for the products of vectors?

M: If the symbol

```
●
```

represents the dot product of two vectors P and Q then

```
●←'+/ω*2' ⍒ '+/α×ω'
```

The monadic form is the dot product of a vector with itself.

The dot product is commutative.

```
'TRUE'⊢(P●Q)≡(Q●P)
```

Dot distributes over addition.

```
'TRUE'⊢(P●(Q+R))≡((P●Q)+(P●R))
```

Multiplication of a dot product by a scalar is associative.

```
'TRUE'⊢(M×(P●Q))≡((M×P)●Q)
```

Further, if the symbol

```
X
```

represents the cross product of two three-element vectors, then

```
X←'' ∇ '((1⌽α)×¯1⌽ω)-(¯1⌽α)×1⌽ω'
```

The cross product is anticommutative.

```
'TRUE'⊢(PXQ)≢(QXP)
```

Cross distributes over addition.

```
'TRUE'⊢(PX(Q+R))≡((PXQ)+(PXR))
```

Multiplication of a cross product by a scalar is associative.

```
'TRUE'⊢(M×(PXQ))≡((M×P)XQ)
```

Dot product and cross product both apply to three dimensions and to arrays whose rank is greater than or equal to one.

S: What about the angle of the vector P with regard to the axes?

M: If the symbol

```
∠
```

called corner, represents a monadic function giving the direction in radians of a vector with respect to the axes, and a dyadic function giving the angle between two vectors.

```
∠←'¯2○ω÷►ω' ∇ '¯2○(α●ω)÷(►α)×►ω'
```

The angle between vectors P and Q is given by

```
P∠Q
```

and the direction of P by

```
∠P
```

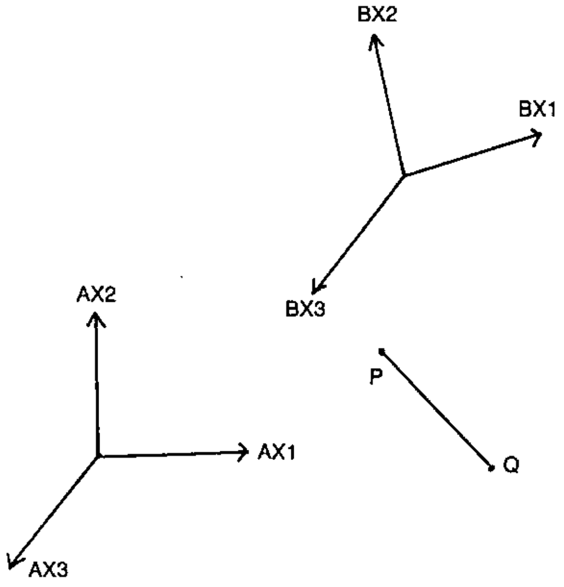
S: It is useful to note that

```
P►Q
```

will give the same result in all stationary frames of reference.

M: Ah! We have a scientific law here. Given any two frames of reference A̲ and B̲ and any two points P and Q

```
A̲P←A̲X1P,A̲X2P,A̲X3P
A̲Q←A̲X1Q,A̲X2Q,A̲X3Q
B̲P←B̲X1P,B̲X2P,B̲X3P
B̲Q←B̲X1Q,B̲X2Q,B̲X3Q
```



```
'LAW'⊢(A̲P▸A̲Q)≡(B̲P▸B̲Q)
```

S: Humm. Before a general statement about nature is raised to the status of a law, it should be subjected to rigorous experimental verification under a variety of circumstances. Given two frames of reference A and B if A were in motion relative to B I am not certain that measurements of the distance PQ in each frame of reference would agree.

M: Don't be a Cretan Socrates! It's obvious they would.

S: Things which seem obvious are not necessarily true, neither is what is true necessarily obvious. Let us think carefully about this. By allowing A to be in motion relative to B we have implicitly introduced the concept of time, for speed is distance moved divided by time taken.

M: Granted.

S: Our judgements of the moment of time (as opposed to chronological order) are actually judgements of simultaneous events. For example, you arrived simultaneously with the shadow on my sun dial here falling at 12 o'clock. While this definition of time is satisfactory for events in the vicinity of my dial, it isn't when we wish to evaluate times of remote events.

M: Why not Socrates?

S: Because it takes time for information regarding these remote events to get here. No signal in nature travels infinitely fast.

You say Achilles found the speed of light to be 871,126,392 mph in all reference frames even if they are moving with respect to one another.

M: Yes he did.

S: Let us use this fact to synchronise a set of clocks in reference frame A using huge measuring rods each of length C miles. If I place identical clockwork mechanisms at each end of a rod I can synchronise them in the following manner. Arrange for a light beam to leave the first clock at 12 o'clock. When the tip of the beam arrives at the second clock, that clock is to be set to 1 o'clock since it takes one hour for light to travel the length of the rod. Let this be done for a grid of clocks separated by such rods.

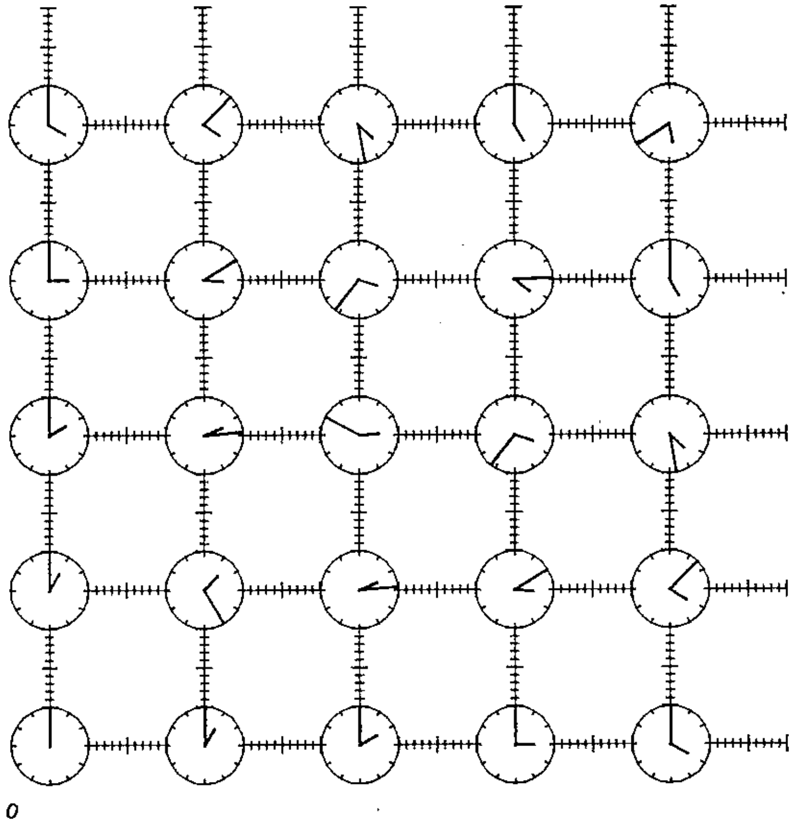
We then have a synchronised space-time reference frame wherein everyone will agree about spacial and temporal assignments. Any event in this reference frame can be given a coordinate position by means of the measuring rods, and a time by associating the moment of the event with the time on the nearest clock. This is the time of the event in that reference frame. Of course, to be accurate, we should have a much finer grid.

At 12 o'clock on our local clock, we believe all other clocks in this synchronised frame will, at that instant, also read 12 o'clock, although, of course, we can't see what they are actually reading at that instant because of the time delay associated with the distance involved.

The stars which we see in the heavens are their younger appearances of even aeons ago!

M: What you say makes syllogistic sense. I can't fault the logic. But do we really need to bother with all these clocks and rods?

In your synchronised space-time reference frame, these are the times at which observers at each clock see the origin clock reading 12.



S: We are trying to find a law of distance which is not affected whether we use one or other of two systems of coordinates in uniform translatory relative motion.

M: Go on then master.

S: Imagine there are two similarly oriented synchronised space-time reference frames A and B and that B is moving at velocity ABX1V in the X1 direction relative to A.

Suppose also that at the moment when their origins meet, both A's and B's origin clocks read 12. At this instant a candle is uncovered at the common origin. Two observers, one at the origin of frame A and the other at the origin of frame B agree that the light departed at 12 o'clock. A second observer in frame B is naturally stationary with respect to the first in B but is situated vertically above him. The distance in miles between them is given by C times the decimal hour on his clock, BT, when the light arrives.

While the light is travelling along axis BX2, the whole reference frame B is moving relative to A. If, when the second observer in B receives the light signal, there happens to be a second observer in A at that position, the question is: what is the reading on his clock?

M: There are four observers altogether. Two in A̲ and two in B̲. One in A̲ and one in B̲ are coincident when the light departs, and one in A̲ and one in B̲ are coincident when the light arrives.

Let's assume decimal clocks at the common origin which both read zero when the light departs. The second clock in A̲ reads A̲T and the second clock in B̲ reads B̲T at a place and time the light arrives. According to A̲ the B̲ clock has moved a horizontal distance

```
    A̲B̲X1V×A̲T
```

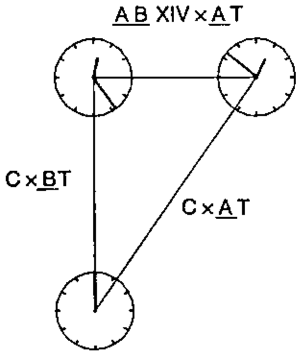
and the light has travelled a diagonal distance

```
    C×A̲T
```

while for B̲ the light has travelled a vertical distance

```
    C×B̲T
```

The situation can be summarised in a right angled triangle.



Applying Pythagoras to this triangle gives the relation between the readings on the clocks in A̲ and B̲ as

```
    ⌈←'÷0○(⊢⍵)÷C' ∇ ''

    'FACT'⊢A̲T=B̲T×⌈(A̲B̲X1V,0,0)
```

I admit I am surprised that clocks at the same place read differently. Surely it is just an illusion that time can go at different rates in different frames of reference. If Castor were to move away from Pollux, they would no longer be twins. If Castor were to turn and come back, since

```
'TRUE'⊢(⌈ω)=(⌈-ω)
```

their clocks would be different when they next met!

Achilles did say his clock kept going slow with respect to Athens mean-time; and he does maintain a youthful appearance!

S: Although from our frame of reference it takes 28 years for light to arrive from Vega (and so we see Vega as it was 28 years ago), a nymph can fly there and only age a few hours without travelling faster than light.

A twin is not absolutely twinned, neither is a journey conclusively small or great!

M: Eh?

S: Considering distances rather than times:

```
A̲L←C×A̲T
B̲L←C×B̲T
'FACT'⊢A̲L=B̲L×⌈(A̲B̲X1V,0,0)
```

So distances too are measurably different in the two frames of reference.

M: Oh I see!

S: Take another example. Again we have the two frames A̲ and B̲ moving at velocity A̲B̲X1V relative to one another in the X1 direction. Again the candle is uncovered when the space-time origins coincide. Since the speed of light is the same when measured in both frames of reference, observers situated at some distance round their own origins will report that the light from their origin forms an expanding sphere centred on their own origin.

M: That seems paradoxical. Surely there is only one sphere. It must be centred either on A̲'s origin or on B̲'s. At any rate, it can't be centred on both because they only coincide at the instant when the light first departs and not thereafter.

S: Consider the situation either from A̲'s point of view or from B̲'s. There is no third absolute perspective.

M: Right. Taking the situation from A̲'s point of view; after time A̲TP the wavefront at some arbitrary point P will have reached a distance C times A̲TP from the origin. In other words, the wavefront will form an expanding sphere radius C times A̲TP at time A̲TP.

```
A̲P←A̲X1P,A̲X2P,A̲X3P
```

A̲P is a point on the wavefront at time A̲TP.

```
'FACT'⊢(C×A̲TP)=▶A̲P
```

S: If you write

```
A̲X4P←0J1×C×A̲TP
A̲P←A̲X1P,A̲X2P,A̲X3P,A̲X4P
```

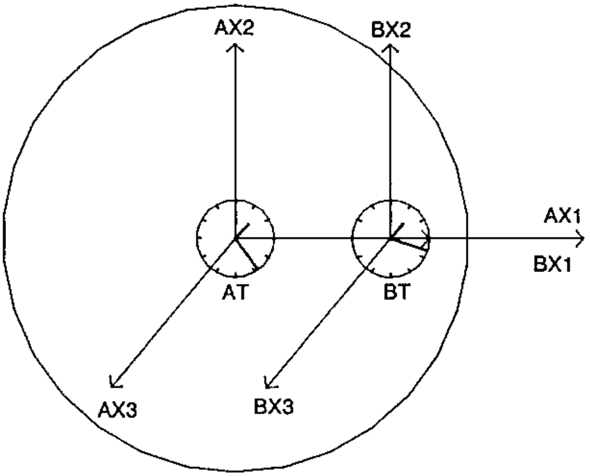
Then the expression becomes

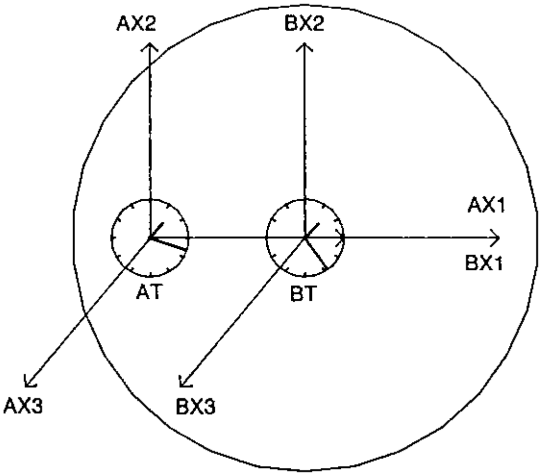
```
'FACT'⊢0=►A̲P
```

M: Nice one master. Exactly the same holds for B̲.

```
B̲X4P←0J1×C×B̲TP
B̲P←B̲X1P,B̲X2P,B̲X3P,B̲X4P
'FACT'⊢0=►B̲P
```

Squaring the OJ1, as is done inside the function Line, gives a factor minus 1; One is then taking the square root of something which is zero giving the length of the four element vector B̲P to be zero for all space-time points on the wavefront.





S: Since the clocks in A̲ and B̲ were synchronized in a manner exactly similar to the arrangement just described, we can now state the law which we originally sought. The four dimensional generalization of Pythagoras' theorem yields; the space-time length from the common origin to *any* point P is the same for all reference frames in uniform relative motion.

```
'LAW'⊢(▸A̲P)=(▸B̲P)
```

Also, for two points P and Q, the four dimensional distance between them is the same for all such frames.

```
'LAW'⊢(A̲P▸A̲Q)=(B̲P▸B̲Q)
```

M: Amazing!

S: Both coordinate systems are describing the same light wave, so there must be a transformation which would take us from A̲ coordinates to B̲ coordinates and back again at will.

M: This transformation must be a function of A̲ B̲X1V. That is the only variable there seems to be here.

S: Carry on Meno.

M: Well, we need a matrix function of a vector, call it 4 Frame, using the symbol ⌼ which takes an argument of the relative vector velocity V (in our case A̲ B̲X1V,O,O) and returns a matrix of four rows and four columns such that

```
'FACT'⊢B̲P≡̲(⌼V)+.×A̲P
'FACT'⊢A̲P≡̲(⌼-V)+.×B̲P
```

This is the case whatever the relative velocity, V. Combining the transformation with its inverse leads to

```
'TRUE'⊢(4 4ρ1 0 0 0 0)≡̲(⌹⌼V)+.×⌼V
'TRUE'⊢(4 4ρ1 0 0 0 0)≡̲(⌼V)+.×⌹⌼V
```

for all V, by definition. These relations lead to the conclusion that

```
'FACT'⊢(⌼-V)≡̲⌹⌼V
```

S: Another way of picturing the relationship between A̲ and B̲ ignoring the X2 and X3 axes because they are not involved, is obtained by regarding the velocity difference between them as a rotation α of the space-time coordinates so that

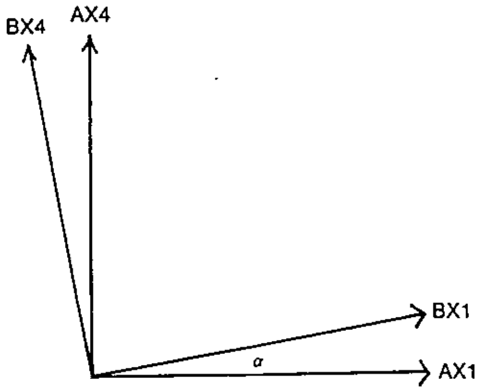
```
'FACT'⊢α=A̲X1ƗB̲X1
```

and then, naturally,

```
'FACT'⊢α=⍙X4∠B̲X4
```

because we are dealing with orthogonal axes.

M: Ah!



Now I can use Euclid[6] to find out what the transformation is. Take an arbitrary point P in the plane and drop perpendiculars from P to both sets of axes.

For a right angled triangle, one of whose adjacent sides is α.

```
1○α
```

is equal to the length of the side opposite to α divided by the length of the hypotenuse, while

```
2○α
```

is the adjacent side over the hypotenuse. Length OU is

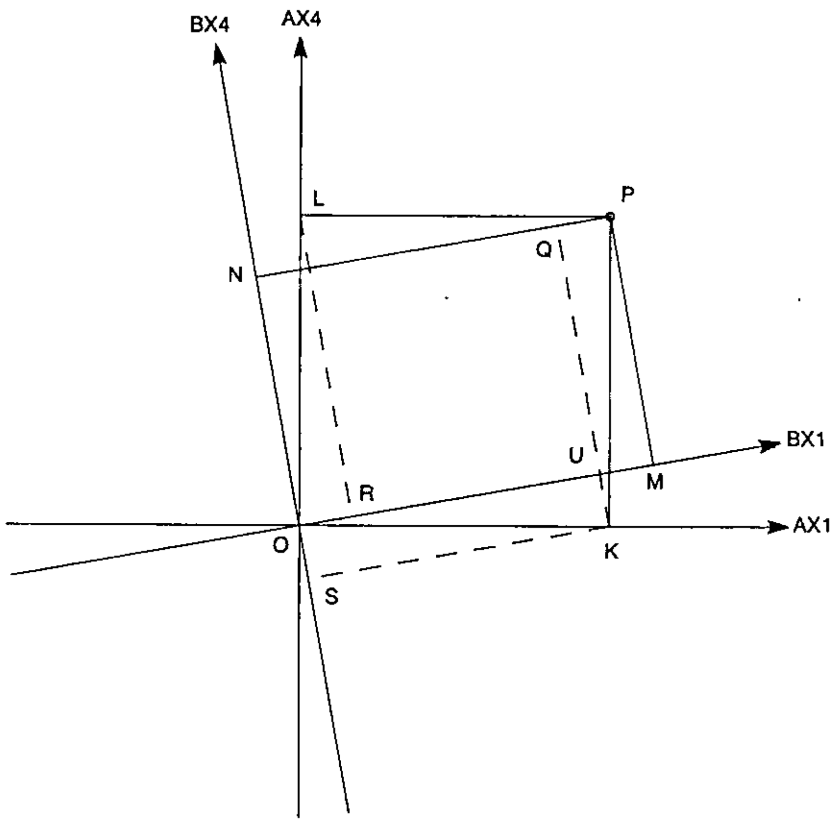
```
(2○α)×⍙X1P
```

since length OK is

```
⍙X1P
```

Length OR is

```
(1○α)×⍙X4P
```



since length OL is

```
A̲X4P
```

Triangle ORL is congruent to triangle KQP since OL is the same length as KP and all the angles are equal. So length OR is the same as QP which is itself the same as UM. Now, OM is length OU plus UM, and OM is length

```
B̲X1P
```

Therefore:

```
'TRUE'⊢B̲X1P=((2○α)×A̲X1P)+((1○α)×A̲X4P)
```

A similar argument gives the expression for

```
B̲X4P
```

Considering that ON is SN minus OS, OS is KU which is

```
    (1○α)×A̲X1P
```

SN is the same distance as QK which is the same as RL by congruence. And RL is

```
    (2○α)×A̲X4P
```

Thus the matrix given by 4 Frame V is formally

```
   2○α      0      0    1○α
    0       1      0     0
    0       0      1     0
  -1○α      0      0    2○α
```

Any point A̲P is related to the same point B̲P by

```
    'TRUE'⊢B̲X1P=((2○α)×A̲X1P)+((1○α)×A̲X4P)
    'TRUE'⊢B̲X2P=A̲X2P
    'TRUE'⊢B̲X3P=A̲X3P
    'TRUE'⊢B̲X4P=((-1○α)×A̲X1P)+((2○α)×A̲X4P)
```

from the geometry.

S: You are well versed in Euclidean geometry Meno. But how are you drawing all these nice pictures?

M: I'm using my new HPQ722123ABC*αω* jotter which I got from the stoneware factory at Sparta. Look!

S: Golly! That is a super machine. What's that beside it?

M: That's my

ζβμ

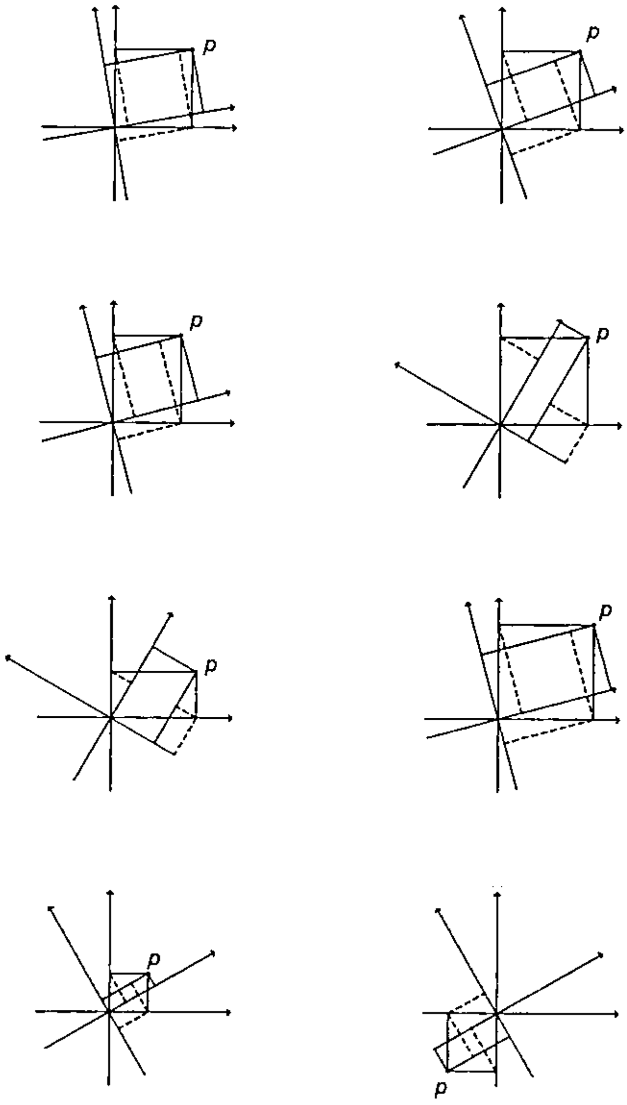
impersonal abacus.

S: I say: have you got Blunt APL on that tiny?

M: O Jupiter! Socrates! You mean Sharp APL on this micro.

S: Er, um .... yes. Do you have Sharp APL on that micro Meno?

M: Of course I do. If you doubt my reasoning proving that OR is the same as UM, or that SN is the same as RL then I can easily write an APL function which takes a left argument of the angle *α* and a right argument of the vector coördinates of P. I can then draw lots of diagrams on this slate with various angles between the axes, and for different points P. You can then see that the lengths of OR and UM as well as SN and RL are always the same.



S: What is *α* then? What is its relation to reality?

M: No slave has a clue master.

S: Can you find it perhaps from

```
'TRUE'⊢B̲X1P=((2○α)×A̲X1P)+((1○α)×A̲X4P)
```

considering that the origin of B̲ is at

```
B̲B̲0←0
```

in frame B̲ and at

```
A̲B̲0←A̲B̲X1V×A̲T
```

in frame A̲.

M: That's crafty Socrates. A̲'s view and B̲'s view of B̲'s spatial origin yield:

```
'FACT'⊢0=((2○α)×A̲B̲X1V×A̲T)+((1○α)×0J1×C×A̲T)
```

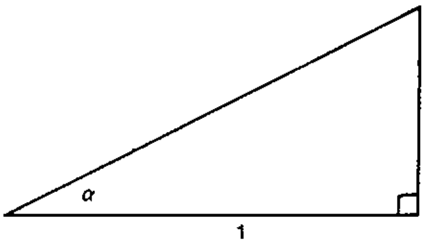
which gives

```
'FACT'⊢(3○α)=0J1×A̲B̲X1V÷C
```

Where do we go from here?

S: Draw a right angled triangle with one angle equal to α and the adjacent side of unit length.

M: Easy.



S: In a right angled triangle, such as you have there, the tangent of α is equal to the opposite side divided by the adjacent side. By construction, the adjacent side has length 1.

M: Ah. I see. We know tan α and the length of the adjacent side, so we can deduce that the opposite side must have length

```
0J1×A̲B̲X1V÷C
```

and therefore the hypotenuse has length

```
⌹1,0J1×A̲B̲X1V÷C
```

which is actually the same as

```
÷⌈A̲B̲X1V
```

Knowing the lengths of the sides of the triangle, expressions for sin and cos of α follow immediately.

```
'FACT'⊢(1○α)=(⌈A̲B̲X1V)×0J1×A̲B̲X1V÷C
'FACT'⊢(2○α)=⌈A̲B̲X1V
```

Replacing these expressions in the transformation and ignoring the X2 and X3 axes which are not involved, gives a matrix function of a scalar,

```
⍁
```

which transforms coordinates

```
A̲P←A̲X1P,A̲X4P
```

to coordinates

```
B̲P←B̲X1P,B̲X4P
```

for any point P when B is applied to velocity ABX1V.

```
⍁←'2 2⍴(⌈⍵)×1,0J1 0J¯1×⍵÷C' ∇ '' ⍤0
```

S: Try it on your abacus. See what happens to B's clocks and rods when viewed from A. Verify that the distance between any two points in space-time is the same in A as in B and hence that

```
'LAW'⊢(▸A̲P)=(▸B̲P)
'LAW'⊢(A̲P▸A̲Q)=(B̲P▸B̲Q)
```

M:

```
      ⍝ TAKE FIRST CLOCK AXIS A̲X1 AT 1 O'CLOCK
      ⎕←A̲1←C×1 0J1
6.7113E8 0J6.7113E8

      ⍝ A̲B̲X1V IS HALF THE SPEED OF LIGHT
      ⎕←A̲B̲X1V←C÷2
335563196

      ⍝ TRANSFORMATION FROM A̲ TO B̲ IS TA̲B̲
      ⎕←TA̲B̲←⍁A̲B̲X1V
1.1547     0J0.57735
0J¯0.57735     1.1547

      ⍝ APPLY TRANSFORMATION TO GET B̲'S VIEW OF A̲1
      ⎕←B̲1←TA̲B̲+.×A̲1
3.8748E8 0J3.8748E8

      ⍝ B̲ SEES THIS CLOCK ALMOST HALF AS FAR FROM THE ORIGIN
      ⍝ AS A̲ SEES IT, AND AT ALMOST HALF THE TIME
      ▸A̲1
0
```

```
      ⍝ A̲1 IS A SPACE-TIME POINT ON THE SYNCHRONISING WAVEFRONT
      ►B̲1
0


      ⍝ B̲1 IS ALSO ON THE SYNCHRONISING WAVEFRONT
      'LAW'⊢(►A̲1)=(►B̲1)
1


      ⍝ TAKE TWO RANDOM POINTS IN SPACE-TIME
      ⎕←A̲P←A̲1×2?1000
3.4784E11 0J4.4294E10
      ►A̲P
3.4481E11
      ⎕←A̲Q←A̲1×2?1000
1.4429E11 0J2.5838E11
      ►A̲Q
0J2.1434E11


      ⍝ THE SPACE-TIME DISTANCE BETWEEN THEM IS:
      A̲P►A̲Q
0J6.6951E10
      ⎕←B̲P←TA̲B̲+.×A̲P
3.7585E11 0J¯1.4957E11
      ►B̲P
3.4481E11
      ⎕←B̲Q←TA̲B̲+.×A̲Q
1.7436E10 0J2.1505E11
      ►B̲Q
0J2.1434E11
      'LAW'⊢(►A̲P)=(►B̲P)
1
      'LAW'⊢(A̲P►A̲Q)=(B̲P►B̲Q)


  ⍝ FOUR DIMENSIONAL DISTANCES ARE THE SAME IN BOTH FRAMES
  ⍝ TAKE A NUMBER OF POINTS
  ⎕←A̲S←C×(⍳3)∘.×1 0J1
       0          0
6.7113E8 0J6.7113E8
1.3423E9 0J1.3423E9


      ⍝ AND A NUMBER OF RELATIVE VELOCITIES
      ⎕←A̲B̲X1VS←C×.3 .5 .9 .99
2.0134E8 3.3556E8 6.0401E8 6.6442E8
      ⎕←TA̲B̲S←⍁A̲B̲X1VS
    1.0483  0J0.31449
0J¯0.31449     1.0483

    1.1547  0J0.57735
0J¯0.57735     1.1547

    2.2942   0J2.0647
 0J¯2.0647     2.2942

    7.0888   0J7.0179
 0J¯7.0179     7.0888
```

```
        ⎕←BS←⍉TABS+.×⍉AS
            0          0          0           0
            0          0          0           0
    4.9247E8   3.8748E8   1.5397E8    4.7575E7
  0J4.9247E8 0J3.8748E8 0J1.5397E8  0J4.7575E7

    9.8494E8   7.7495E8   3.0793E8     9.515E7
  0J9.8494E8 0J7.7495E8 0J3.0793E8   0J9.515E7
```

S: Can I borrow your impersonal abacus for a minute please Meno?

M: Here you are then.

S: Thanks a lot.
'BANG BANG BASH BASH BANG ......'

M: Go easy Socrates, you'll break the keyboard!!

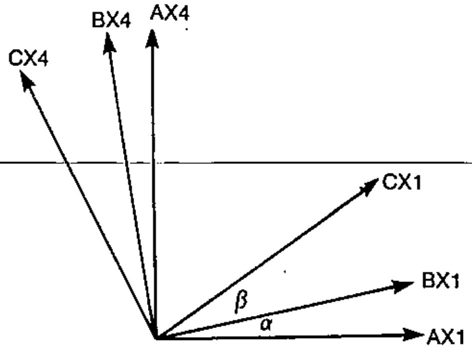
S: Ah yes! I think the four dimensional transformation for a general three dimensional relative velocity motion is given by

```
    ⎕←'((⎕ω),⎕ω),¯⍉(-⎕ω),⌈ω' ∇ ∘⍤1
    ⎕←'(I∘.=I←⍳3)+ω∘.×ω×((⌈ω)-1)÷ω+.×ω' ∇ ∘
    ⎕←'0J1×(⌈ω)×ω÷C' ∇ ∘
```

If there is a third reference frame C moving at velocity BCX1V in the X1 direction with respect to B ....

M: Oh please stop master. Perhaps, after all, the gods from their Olympian frame have an entirely different view from us of why we are in our here and now .....

S: ...... the picture would be this:



The velocity ACX1V of C relative to A is

```
    'FACT'⊢ACX1V=0J¯1×C×3○α+β
```

Since the tangent of the sum of two angles is equal to the sum of the tangents of each angle divided by 1 minus the product of the tangents of each:

```
'FACT'⊢ACX1V=(ABX1V+BCX1V)÷1+(ABX1V×BCX1V)÷C*2
```

which implies that one cannot reach the velocity C by compounding a finite number of relative velocities less than C.

M: SOCRATES!!

S: In summary:

NAME OF SYMBOL	SYMBOL	OVER-STRIKE	MONADIC NAME	RANK	DYADIC NAME	RANK	DEFINITION
LINE	⊷	⊢•	LENGTH	0 1	DISTANCE	0 1 1	'(+/⍵*2)*÷2' ∇ '(+/(⍺-⍵)*2)*÷2'
DOT	●	○•	DOT	0 1	DOT	0 1 1	'+/⍵*2' ∇ '+/⍺×⍵'
CROSS	X	/\			CROSS	1 1 1	'' ∇ '((1⌽⍺)×¯1⌽⍵)-(¯1⌽⍺)×1⌽⍵'
CORNER	⊾	L/	DIRECTION	1 1	ANGLE	0 1 1	'¯2○⍵÷⊷⍵' ∇ '¯2○(⍺●⍵)÷(⊷⍺)×⊷⍵'
GAMMA	Γ	Γ¯	GAMMA	0 1			'÷0○(⊷⍵)÷C' ∇ ''
2FRAME	⍁	2⎕	2FRAME	2 0			'2 2⍴(Γ⍵)×1,0J1 0J¯1×⍵÷C' ∇ ⍎⍕0
1QUAD	⎅	1⎕	1QUAD	1 1			'(Γ⍵)×0J1×⍵÷C' ∇ ''
3QUAD	⍰	3⎕	3QUAD	2 1			'(I∘.=I←⍳3)+⍵∘.×⍵×((Γ⍵)-1)÷⍵+.×⍵' ∇ ''
4FRAME	⌺	◇⎕	4FRAME	2 1			'((⍰⍵),(⎅⍵),¯⍉(-⎅⍵),Γ⍵' ∇ ⍎⍕I

M: O Socrates, now that I know all this I must be wise.

S: O Meno, now that I know all this I know I must be ignorant.

REFERENCES

[1] Socrates (470-399 BC) was condemned to death for misleading the youth of Athens. He taught that it is preferable to suffer wrong than commit it.

[2] Rationalized APL (1983). I.P. Sharp Associates Research Report Number 1 by K. E. Iverson.

[3] Achilles was cited in a paradox by Zeno (490-430 BC) suggesting Achilles could never beat a tortoise in a race if the latter gets a start because always by the time he arrived at the point the tortoise was it has moved on half again, say. It is solved by

```
'TRUE'⊢2=+/÷2*(⍳¯)-⎕IO
```

[4] Pythagoras (570-490 BC) attempted to formulate a mathematical explanation of all reality. He preached the doctrine of the transmigration of souls.

[5] Democritus (480-370 BC) argued that since nothing can come from nothing, and motion, which requires a void, really occurs, reality must consist of atoms moving in a void.

[6] Euclid (323-283 BC), is the famous Greek geometrician.

G. D. Robertson
IPSA Ltd
10 Dean Farrar St
London SW1
01-222-7033

QUOTITIAN JOTTINGS ON MATTERS TEMPORAL

by Michael Carmichael

Part 1 — The "TICTOCTI" — an unfamiliar subdivision of a minute

While browsing through an obscure anthropological journal the other day I chanced upon the following entry:

> 'Professor Julian Day and Dr. Sid E Real (who recently reported from a distant South Pacific island that the natives have 255 different words for COCONUT) have made another startling discovery. In this remote corner of the world the passage of time is measured in hours, minutes and TICTOCTIs. What precisely is a TICTOCTI?
>
> 1 TICTOCTI = 1⅓ TICTOCs = 1.3333... seconds
>
> 'The unfamiliar subdivision of a minute is said to be more in keeping with the pace of life on those sun-drenched coral shores. "Our discovery is of great significance to theologians, philosophers and manufacturers of multi-function digital wrist-watches" the Professor said.'

Having learned of these resourceful and free-thinking islanders I immediately resolved to adopt their curious unit for my 'business micro' projects. As a measure of elapsed time of day it proved to be ideal. The number of TICTOCTIs in 24 hours is 64800 which by happy coincidence is somewhat less than 2 to the power of 16. Thus any instant in a 24 hour period may be conveniently represented as a 2-byte integer by conversion to TICTOCTI units.

The method is straightforward. To pack HH-MM-SS into TICTOCTI units:

```
R←¯32768+⌊0.75×24 60 60⊥A
```

Note the 2*15 offset which keeps the number within the 2-byte limit.

To unpack TICTOCTI units back to HH-MM-SS

```
R←24 60 60⊤⌊(32768+A)÷0.75
```

Experiment confirms the hunch that the method is 'spot on' to the nearest second for 75% of the time and only 'one second out' for the other 25%.

If your APL does not feature 2-byte integers (or your auditors insist on timestamps recorded to the nearest millisecond) then you may find the TICTOCTI a little academic (like the Professor). But for some micro-bound APLers the anthropologists' serendipitous discovery introduces an efficient way to store the time of day, or to put it another way, save space when you save time.

Here are the functions for doing the conversions:

```
    ∇ R←TIMP A
[1]  ⍝ Pack 3-column HH-MM-SS timestamp into 2-byte TICTOCTI units
[2]  ⍝ N.B. 1 TICTOCTI = 1.33333.....secs.
[3]   R←¯32768+⌊0.75× 24 60 60 ⊥⍉A
    ∇

    ∇ R←TIMU A
[1]  ⍝ Unpack 2-byte TICTOCTI units into 3-column HH-MM-SS timestamp
[2]  ⍝ N.B. 1 TICTOCTI = 1.33333.....secs.
[3]   R←⍉ 24 60 60 ⊤1+⌊(32768+A)÷0.75
    ∇
```

Examples:

```
      +A←TIMP 2 3⍴0 0 1,23 59 59
¯32768 32031

      TIMU A
  0  0  1
 23 59 59

      +A←TIMP 1 3⍴3↓⎕TS
13380

      TIMU A
 17  5 31

      ,0 2↓TIMU TIMP 0,0,25 1⍴⍳25
1 2 3 5 5 6 7 9 9 10 11 13 13 14 15 17 17 18 19 21 21 22 23 25 25
```

Part 2 — DDMYY — the cult of the negative month number

Professor Julian Day's disclosure of chronological unorthodoxy in the South Pacific does not end with the 'tictocti' unit. The islanders are pioneers of the NEGATIVE MONTH NUMBER.

Let me lessen the blow by arguing that the assignment of integers 1 to 12 for the set of twelve months January to December is only one dull choice out of a rich set of possibilities. For example:

- The Mathematician's Delight. Proponents of index origin zero might favour the series 0 to 11. This would be consistent with a culture which calls the second level of a building the first floor. Ergo JAN = 0.
- The Historical Perspective. Is this not reasonable:

  SEPT, OCTO, NOVE, DECE = 7, 8, 9,10

  If December is the tenth month (it was once!) then surely JAN = −1.
- The Taxman's Rationale. The Inland Revenue gets down to business in the second quarter of the year. Foolish New Year celebrations take place on April 1st. In this case JAN = −2.

Why our South Sea Islanders should favour the approach of the Inland Revenue must be one of the great mysteries of our time. Unfortunately Prof. Julian Day throws no light on this, but his article (appearing in the latest University of Nether Wallop Bulletin) does supply the following table:

JAN	−2
FEB	−1
MAR	0
APR	1
MAY	2
JUN	3
JUL	4
AUG	5
SEP	6
OCT	7
NOV	8
DEC	9

Did you know that APL2 has a time zone feature quadTZ which can be used to fiddle with the hours of quadTS? Our Antipodean friends have been fiddling with the months as well!

Their format takes some getting used to. Whereas Europeans favour:

DDMMYY

and Americans go for:

MMDDYY

The natives of our South Pacific island insist upon:

DDMYY (SIGNED!)

By another happy (and unbelievable) coincidence, all dates between 1950 and 2049 written in this unconventional format lie within the range −32768 to 32767. The South Pacific fiscal year leads to a practical way of packing a date into a 2-byte integer.

Here are the functions with some examples (bearing in mind that in that far off island years are counted from 1950 — the date they gained independence):

```
      ∇ R←DATP A
 [1]   ⍝ Pack 3-column YY-MM-DD dates in <A> into 2-byte format
 [2]    R←A-((1↑ρA),3)ρ 0 3 1950
 [3]    R←(1-2×0>R[;2])×+/(|R)×(ρR)ρ 1000 100 1
      ∇

      ∇ R←DATU A;MM
 [1]   ⍝ Unpack 2-byte dates in <A> into 3-column YY-MM-DD format
 [2]    R←|A ◇ MM←3+(×A)×10|⌊R÷100
 [3]    R←(⌊R÷1000),MM,[1.5]1950+100|R
      ∇
```

```
      DATP 2 3ρ1 1 1950 , 31 12 2049
¯1200 31999

      ⍝ The Frisbee introduced by Wham-O Manufacturing Co.:
      DATP 1 3ρ13 1 1957          ⍝ 13th January, 1957
¯13207

      ⍝ Boy Scouts permitted to wear long trousers:
      DATP 1 3ρ1 11 1961          ⍝ 1st November, 1961
1811

      ⍝ Women win the right to vote in Switzerland:
      DATP 1 3ρ7 2 1971           ⍝ 7th February, 1971
¯7121

      ⍝ Create some test data:
      DMY←⌽(1949+?1000ρ100),(?1000ρ12),[1.5]?1000ρ28

      ⍝ Check that functions work:
      ∧/,DMY=DATU DATP DMY
1

      ⍝ Compare with APL*plus/PC supplied functions
      ⍝ from workspace <1 DATES> :
      ∧/,DMY=⌽DATEREP DATEBASE⌽DMY
1

      ⍝ Timings show that <DATP> and <DATU> are almost 3 and 4
      ⍝ times faster than <DATEBASE> and <DATEREP>, respectively,
      ⍝ using APL*plus/PC release 3.1 on an IBM PC with an 8087
      ⍝ maths co-processor installed.
```

If YYMMDD is unattractive in APL*Plus/PC what is wrong with a day number solution? Day numbers are excellent for sorting, date-differencing, days of the week etc. But conversion to and fro is quite complex and therefore slow. The 'taxman's date' described above is 4 times quicker to unpack than a day-number to date conversion. Which prompts the question: Are day numbers' days numbered?

RANDOM CONTINGENCY TABLES : A USE FOR '3-DIMENSIONAL MATRICES'

by Dr. A. G. Prys Williams

A contingency table is a 2-dimensional array of positive numbers together with their row and column sums and the overall total, such as

3009	2832	3008	8849
3047	3051	2992	9095
2974	3038	3018	9030
9030	8921	9023	26974

An obvious example is a spread-sheet of sales figures. A standard problem in simulation is that of finding random figures which have the same row and column totals as the originals; for instance, for finding possible sales figures that would have given the same overall results. From this one can tell whether an outstandingly high, or low, figure could reasonably be due to chance. In APL terms, this involves finding, for an RxC dimensional original, an NxRxC array of integers with the right marginal totals.

We tackle this problem according to two sound APL principles:

i) Work with arrays, not loops, where possible.

ii) Use existing idioms where possible.

The existing idiom for n-dimensional arrays of independent random normal deviates is well developed, e.g.

```
      ∇ K←NRAN R;N;Z
[1]     K←(¯0.5+?(Z←2,⌈(N←×/,R)÷2)⍴10000000)÷10000000 ⍝ BOX-MULLER SIM.
[2]     K←R⍴N↑((¯2×⍟(×/Z)⍴K[1;])*0.5)×, 1 2 ∘.○2×K[2;] ⍝ R-SHAPED OUTPUT
      ∇
```

so we use a downright ancient technique to relate random contingency tables to random normals, drawing on the linked papers of Irwin (1949) and Lancaster (1949).

The condition of having fixed row and column totals can be expressed by using fixed row (or column) probabilities $p_1 \ldots p_n$ where p_i will be the ith row total divided by the grand total, and so on. The expected frequencies in the ith row and jth column will be e_{ij} = grand total x row probability x column probability, and if the observed count in that cell is n_{ij} then $(n_{ij} - e_{ij}) / \sqrt{e_{ij}} = q_{ij}$ gives a matrix Q of standardised deviations from the expected frequencies.

In the numerical example, this is

$$Q = \begin{bmatrix} 0.857 & -1.749 & 0.880 \\ 0.042 & 0.785 & -0.822 \\ -0.890 & 0.943 & -0.047 \end{bmatrix}$$

The known row or column probabilities can be used to construct a pair of matrices using the formula

$$l_{ij} = \sqrt{(p_j)}\ (j = 1,2,\ldots,n),$$

$$l_{ij} = \sqrt{\left\{p_i p_j \Big/ \left[\left({}_{t=1}^{i-1}p_t\right)\left({}_{t=1}^{i}p_t\right)\right]\right\}}\ (i = 2,3\ldots,n; j = 1,2\ldots i-1),$$

$$l_{ij} = -\sqrt{\left({}_{t=1}^{i-1}p_t \Big/ {}_{t=1}^{i}p_t\right)}\ (j = i).$$

$$l_{ij} = 0\ (i = 2,3\ldots,n; j = i + 2,\ldots n).$$

for the row probabilities to get

$$R = \begin{bmatrix} 0.573 & 0.581 & 0.579 \\ 0.712 & -0.702 & -0 \\ 0.406 & 0.412 & -0.816 \end{bmatrix}$$

$$C = \begin{bmatrix} 0.578 & 0.575 & 0.578 \\ 0.705 & -0.709 & -0 \\ 0.410 & 0.408 & -0.816 \end{bmatrix}$$

These matrices are orthogonal, that is, with their inverse equal to their transpose. They are calculated by the APL function

```
    ∇ R←ORTHOG P;I;J;N;CP
      CP←+\P ⍝      HELMERT TRANSFORMATION MATRICES NEEDED FOR <RANTAB>.
      N←⍴P ⍝        COPIED DIRECTLY FROM: IRWIN, 'A NOTE ON THE
      R←(N,N)⍴0 ⍝   SUB-DIVISION OF CHI-SQUARED INTO COMPONENTS',
      R[1;]←P*0.5 ⍝ BIOMETRIKA, 1949, PP. 130-4.
      I←2
    K:J←1 ⍝        <P> IS A VECTOR OF ROW OR COLUMN PROBABILITIES.
      R[I;I]←-(CP[I-1]÷CP[I])*0.5
    L:R[I;J]←(P[I]×P[J]÷(CP[I]×CP[I-1]))*0.5
      →((J←J+1)<I)/L
      →((I←I+1)≤N)/K
    ∇
```

which is an exact transcription.

We then find that

$$Z = RQC^T$$

has zeros along the first row and column and independent random normal deviates elsewhere, e.g.

$$Z = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1.683 & -1.477 \\ 0 & 1.590 & -0.071 \end{bmatrix}$$

If we generate say 1,000 such sets of random normal deviates and reverse the transformation, we will get 1,000 sets of numbers with the right row totals, column totals, and statistical independence properties. If we round them to whole numbers, our task is done. Hence

```
     ∇ Z←NS RANTAB RC;ROW;COL;TOT;R;C;E;N;A;B
[1]   ⍝ SIMULATION OF RANDOM R×C CONTINGENCY TABLES.
[2]   ⍝ FOR METHOD SEE LANCASTER, 'THE DERIVATION AND PARTITION OF
[3]   ⍝ CHI-SQUARE IN CERTAIN DISCRETE DITRIBUTIONS',
[4]   ⍝ BIOMETRIKA, 1949, PP117-29.
[5]    N←1↑NS ⍝      NUMBER OF SIMULATIONS REQUIRED
[6]    A←NS[2] ⍝     NUMBER OF ROWS
[7]    B←NS[3] ⍝     NUMBER OF COLUMNS
[8]    ROW←A↑RC ⍝    ROW TOTALS
[9]    COL←(-B)↑RC ⍝ COLUMN TOTALS
[10]   TOT←+/ROW
[11]   →(TOT=+/COL)/C̲ ⍝ CHECK CONSISTENCY OF SPECIFICATIONS
[12]   'OVERALL TOTALS INCONSISTENT'
[13]   →0
[14]  C̲:R←ORTHOG ROW÷TOT
[15]   C←ORTHOG COL÷TOT
[16]   E←ROW∘.×COL÷TOT
[17]   Z←(2 3 1 ⍉(⍉R)+.×0,[2]0,[1]NRAN(A-1),(B-1),N)+.×C ⍝ SIMULATION
[18]   Z←⌊0.5+E+((E←((N,(A-1),(B-1))⍴ ¯1 ¯1 ↓E))*0.5)× 0 ¯1 ¯1 ↓Z ⍝ ROUND
[19]   Z←Z,[2]((N,B)⍴COL)-+/[2]Z←Z,[3]((N,(A-1))⍴¯1↓ROW)-+/[3]Z ⍝ FILL
[20]   Z←(E←∧/[2]∧/[3]Z≥0)/[1]Z ⍝ CHECK FOR NEGATIVE FREQUENCIES
[21]   →(N>+/E)×E̲ ⍝ STOP IF FREQUENCIES ALL POSITIVE, OTHERWISE
[22]  E̲:Z←Z,[1]((N-+/E),A,B)RANTAB RC ⍝ GO BACK FOR MORE
     ∇
```

Breaking this down, lines 5 to 13 split the first parameter up into the dimensions of the result NxAxB and the second parameter up into the required row and column totals.

```
[5]    N←1↑NS
[6]    A←NS[2]
[7]    B←NS[3]
[8]    ROW←A↑RC
[9]    COL←(-B)↑RC
[10]   TOT←+/ROW
[11]   →(TOT=+/COL)/C̲
[12]   'OVERALL TOTALS INCONSISTENT'
[13]   →0
```

The next three lines

```
[14]  C̲:R←ORTHOG ROW÷TOT
[15]   C←ORTHOG COL÷TOT
[16]   E←ROW∘.×COL÷TOT
```

generate the non-random matrices E, R, and C Notice that ORTHOG only has to be used twice, however many simulations are required, so that the presence of loops in this function does not much matter.

The generation of Nx(A – 1) × (B – 1) independent random normals is, of course, a single call on the function NRAN.

The '3-dimensional matrix' Z that results when zeros are glued on in the first row and column.

```
Z←0,[2]0,[1]NRAN(A-1),(B-1),N
```

is of dimension AxBxN in order that it may be pre-multiplied by the AxA matrix R^T. It is then converted to the intended form by

```
Z←2 3 1⍉Z
```

and post-multiplied by the BxB matrix C , to get a 3-dimensional form of the equation

$$Q = R^T ZC$$

If we did not have to round the result to whole numbers, we could then take

```
Z←E+((E←(N,A,B)⍴E)*.5)×Z
```

As it is, we drop the last row and column and round everything that is left to the nearest whole number, giving

```
Z←⌊0.5+E+((E←((N,(A-1),(B-1))⍴¯1 ¯1↓E))*0.5)×0 ¯1 ¯1↓Z
```

The compound of all this is lines 17 and 18

```
[17]  Z←(2 3 1 ⍉(⍉R)+.×0,[2]0,[1]NRAN(A-1),(B-1),N)+.×C
[18]  Z←⌊0.5+E+((E←((N,(A-1),(B-1))⍴ ¯1 ¯1 ↓E))*0.5)× 0 ¯1 ¯1 ↓Z
```

It is now a simple matter to fix the row and column totals once and for all in line 19. Occasionally, this may give rise to a negative entry, which is detected in line 20 and edited out of the list of random tables. If this happens, line 21 sends the function back for more in line 22.

```
[19]  Z←Z,[2]((N,B)⍴COL)-+/[2]Z←Z,[3]((N,(A-1))⍴¯1↓ROW)-+/[3]Z
[20]  Z←(E←∧/[2]∧/[3]Z≥0)/[1]Z
[21]  →(N>+/E)×E
[22] E:Z←Z,[1]((N-+/E),A,B)RANTAB RC
    ∇
```

Unless the expected frequencies are very small, it is unlikely that line 22 will be activated. Thus the more elegant recursive approach has been preferred to a loop, despite the need to repeat lines 5-16.

For any heretics who dislike the idea of '3-dimensional matrices' and would prefer a loop, an alternative version is offered, with the rounding procedure hived off to a separate function.

```
        ∇ O←ROW RTABLE1 COL;TOT;R;C;EXP;N;A
   [1]    'INPUT NUMBER OF CYCLES'
   [2]    N←⎕
   [3]    O←(0,(⍴ROW),(⍴COL))⍴0
   [4]    TOT←+/ROW
   [5]    →(TOT=+/COL)/C
   [6]    'OVERALL TOTALS INCONSISTENT'
   [7]    →0
   [8]   C:R←ORTHOG ROW÷TOT
   [9]    C←ORTHOG COL÷TOT
   [10]   EXP←ROW∘.×COL÷TOT
   [11] S:A←NRAN((⍴ROW)-1),((⍴COL)-1)
   [12]   A←0,[1]0,A
   [13]   A←(⍉R)+.×A+.×C
   [14]   A←EXP+A×EXP*0.5
   [15]   A←ROUNDB A
   [16]   →(⎕←0<(∨/∨/(A<0)))/S
   [17]   O←O,[1]A
   [18]   →((N←N-1)>0)/S
        ∇

        ∇ R←ROUNDB A
   [1]    R← ¯1 ¯1 ↓⌊A+0.5
   [2]    R←R,(¯1↓+/A)-+/R
   [3]    R←R,[1](+⌿A)-+⌿R
        ∇
```

A quad in line 16 is to keep a terminal awake (this version is rather slow) and is entirely optional.

The nearest equivalent program in FORTRAN is explained and listed in Patefield (1981). It would, however, be very difficult to use it as in the following example.

Example

```
      ⍝ RESULTS OF SALES FROM TOWNS A THROUGH H ARE AS FOLLOWS:

TOWN:             A    B    C    D    E    F    G    H
COUNCIL SALES:   70  380  201  157  202  208  262  370 |  1850
DOMESTIC SALES:  80  114  215  130  163  186  310  201 |  1399
                 --------------------------------------|-------
                150  494  416  287  365  394  572  571 |  3249

      ⍝ QUESTION: IS TOWN B RELYING TOO MUCH ON COUNCIL BUYING?

      Y←(100 2 8 RANTAB ROWS,COLS)[;1;2]
      ⌈/Y←Y[⍋Y]
306

      ⍝ THE ANSWER WOULD APPEAR TO BE 'YES'
      ⍝ TO SEE WHAT THE UPPER STATISTICAL LIMITS ARE:

      Y[90 95 99 100]
293 295 304 306

      ⍝ JUST TO MAKE ABSOLUTELY SURE:

      ⌈/Y←(100 2 8 RANTAB ROWS,COLS)[;1;2]
305
      ⌈/Y←(1000 2 8 RANTAB ROWS,COLS)[;1;2]
320

      ⍝ A SECOND RUN PRODUCES THE SAME ANSWER AS THE FIRST.
      ⍝ A RUN OF 1,000 TRIALS PUSHES THE LIMIT UP A BIT,
      ⍝ BUT TOWN B IS STILL OVER THE TOP.
```

REFERENCES

Irwin, J.O. (1949), A Note on the Subdivision of X^2 into Components. **Biometrika** 36, 130.

Lancaster, H.O. (1949), The Derivation and Partition of X^2 in Certain Discrete Distributions. **Biometrika** 36 117.

Patefield W. M. (1981) AS 159: An efficient method of generating random RxC tables with given row and column totals. **Applied Statistics** 30 pp. 91-97.

Dr. A.G. Prys Williams
Management Science of Statistics
University College of Swansea
SWANSEA SA2 8PP

INDEX TO ADVERTISERS

APL People Ltd	Page 13
Cocking & Drury	Page 82
Dyadic Systems Ltd	Page 20
E & S Associates	Page 125
MicroAPL Ltd.	Page 26
I.P. Sharp Associates	Page 90
Metapraxis	Page 112
Metatechnics Systems Ltd	Pages 4 & 46

Potential advertisers are reminded that advertisements for the next issue of VECTOR should be booked by the third Friday in April. For further details please contact David Preedy at Metapraxis Ltd., Hanover House, Coombe Road, Kingston-upon-Thames, KT2 7AH, or phone 01-541-1696.

Please note the following copy dates for future issues of VECTOR:

VECTOR Vol.2 No. 1 ..... May 31st
VECTOR Vol.2 No. 2 ..... Aug 31st
VECTOR Vol.2 No. 3 ..... Nov 30th
VECTOR Vol.2 No. 4 ..... Feb 29th

The editors would be delighted to receive future contributions on floppy disk. This greatly simplifies the job of proof-reading, and cuts our typesetting costs considerably.

The preferred format is:

IBM PC .... DOS 2.0 or later

Please save the file unformatted (or print it to disk) so that there are no control characters embedded in it.

Please also include a hardcopy version, so that we can mark it up sensibly for the printers.

We shall of course return you disks!

Thanks.

BRITISH APL ASSOCIATION

Membership Application Form

Please read the membership information in the inside front cover of VECTOR before completing this form. Existing members who have not already done so should send in an application to update our records; this will be credited pro-rata for any advance membership fees already paid. Use photocopies of this form for multiple applications. The membership year runs from 1st May 1984 — 30th April 1985.

Name:	______
Department:	______
Organisation:	______
Address line 1:	______
Address line 2:	______
Address line 3:	______
Address line 4:	______
Post or zip code:	______
Country:	______
Telephone number:	______

Membership category applied for (tick one):

Non-voting student membership	Free
UK private membership	£ 6
Overseas private membership......	£ 10
Corporate membership	£ 50
Sustaining membership	£250

For student applicants:

Name of course:	______
Name and title of supervisor:	______
Signature of supervisor:	______

PAYMENT

Payment should be enclosed with membership applications in the form of a UK sterling cheque or postal order made payable to "The British APL Association". Corporate or sustaining member applicants should contact the Treasurer in advance if an invoice is required.

Send the completed form to the Treasurer at this address:
Mel Chapman, 12 Garden Street, Stafford, ST17 4BT, UK.

THE BRITISH APL ASSOCIATION

The British APL Association is a Specialist Group of the British Computer Society and a member of EuroAPL, an organisation supported by the Commission of the European Communities. It is administered by a Committee of eight officers who are elected by the vote of Association members at 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.

Photoset by Wizard Printing Services, Brookwood, Surrey.
Printed in England by Copyprint Ltd., London SW1.

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 and a member of EuroAPL, an organisation supported by the Commission of the European Communities. APL stands for "A Programming Language" — and interactive computer programming language noted for its elegance, conciseness and fast development speed. It is supported on many timesharing bureaux and on most mainframe, mini and micro computers.

SUSTAINING MEMBERS

The Committee of the British APL Association wish to acknowledge the generous financial support of the following Association Sustaining Members. In many cases these organisations also provide manpower and administrative assistance to the Association at their own cost.

APL People	3 Cross Lane Close, Orwell, Nr. Royston, Herts. SG8 5QW Tel. 04626-77375
APL*Plus Ltd.	Aston Science Park, Love Lane, Birmingham B7 4BJ. Tel. 021-359 5096
CACI Inc. International	Oriel House, 26 The Quadrant, Richmond, Surrey TW9 1DL. Tel. 01-940 3606
Cocking & Drury Ltd.	16 Berkeley Street, London W1X 5AE. Tel. 01-493 6172
Dyadic Systems Ltd.	30 Camp Road, Farnborough, Hants. Tel. 0252 547222
Inner Product Ltd.	Eagle House, 73 Clapham Common Southside, London SW4 9DG. Tel. 01-673 3354
MicroAPL Limited	Unit 1F, Nine Elms Industrial Estate, 87 Kirtling Street, London SW8 5BP. Tel. 01-622 0395
I.P. Sharp Associates	10 Dean Farrar Street, London SW1H 0DX. Tel. 01-222 7033

The British Computer Society, 13 Mansfield Street, London W1M 0BD.