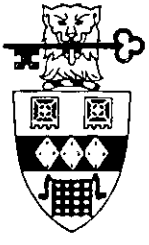
VECTOR

INAUGURAL ISSUE

- Inside: UK and International APL News and Views.
- Spotlight on graphics and information centres.
- Communications: We take the work out of networking.
- Fifth generation language – the paper they dared not print.
- Technical Forum: Complex numbers, and Boole rules OK.
- New competition and over 150 pages of APL action.



The Journal of the British APL Association

A Specialist Group of the British Computer Society

Vol.1 No.1 May 1984

IMPORTANT: Join the British APL Association now to continue receiving VECTOR. [illegible]

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

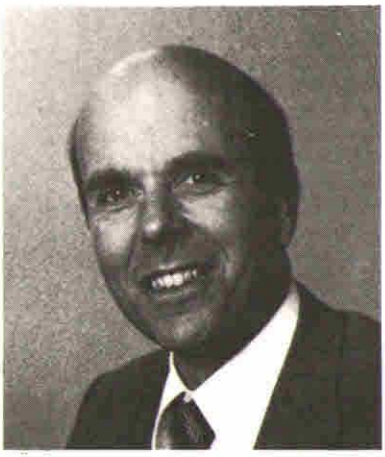
Advertisements in VECTOR should be submitted in typeset camera-ready A5 portrait format with a 20 mm blank border. Illustrations should be black-and-white photographs or line drawings. Rates are £200 per page. A6 and A7 sizes are offered pro-rata subject to layout constraints. Deadlines are:

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

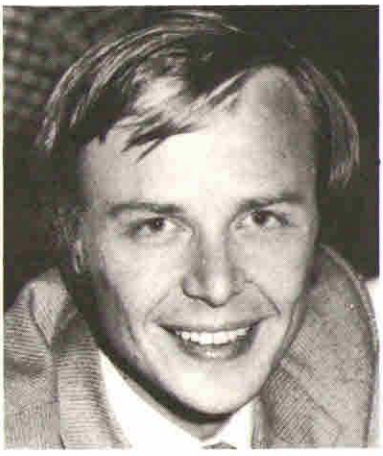
CONTENTS

The VECTOR Working Group

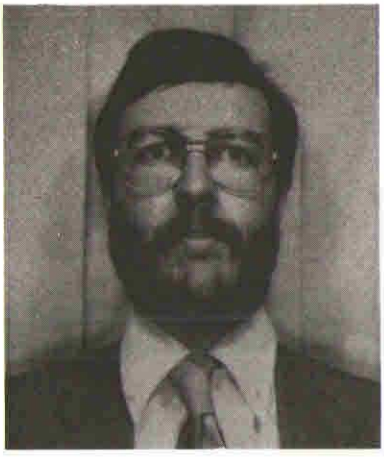
Jonathan Barman
Technical Section



Robert Bittlestone
Editor



David Preedy
Advertising & Reviews



David Ziemann
Technical Section

Adrian Smith is also a valued member of the editorial team but unfortunately his photograph failed to meet the print deadline

EDITORIAL: Back to BASICs?

by Robert Bittlestone

This is the first issue of VECTOR: a new journal for a new generation of computer users. Although VECTOR is aimed at those who know and use APL, it's also very much intended for those who don't yet but might. So we make no apology for including articles which look at APL from the viewpoint of a non-APLer — including that of the general manager who'd like to know why APL makes sense without necessarily learning it personally.

APL is different from all other computer languages in many ways. These differences are part of what gives the language its strength, but also part of what causes it to be criticised. One of the most interesting points about its users is that there are virtually no cases of reverse conversion. Once you become committed to APL, it seems that you are rarely lured away from it. This kind of loyalty is praiseworthy indeed but also has a dangerous tendency to promote insularity. Events might just be passing APL by.

It's almost as if there are two kinds of computing. There is the world out there of COBOL and FORTRAN and BASIC and everything else. This world has its bad points — very slow project development and severe design regimentation being two of them. However, it also has its good points, which include a civilised approach to word processing, imaginative use of on-screen colour graphics, the advent of new user-friendly devices such as the "mouse", the use of a keyboard and character set that is well-nigh universal, and language availability on almost all computers.

Then there is the world of APL. Here we find some bad points too — extremists who confuse complexity with clarity, dreadful legends of undocumented and unsupportable systems, and obscure non-portable techniques required to cope with standard problems such as lower-case output. However, we also find a great many good points, such as elegant structures, powerful and appropriate array handling, flexible and sophisticated intrinsic features and a relational mentality that accelerates design as well as implementation.

I hope I have by now established the notion that VECTOR is not going to be a sycophantic journal. We aim to stimulate by criticism as well as by praise. If we sometimes infuriate, then be sure it is for a purpose. Write us a letter to object: but above all, keep the dialogue going. We have in our hands perhaps the most powerful tool yet devised for harnessing the power of the computer for human needs. So please let's agree to concentrate on honing its cutting edge, rather than laying out rococo designs for its handle.

The production of VECTOR is very much a team effort and I would like to acknowledge the contributions of the journal working group. Jonathan Barman and David Ziemann have painstakingly assembled all the technical content of the journal and vetted contributions for accuracy. David Preedy has worked miracles in producing a new product guide and in encouraging advertisers and sustaining members. Adrian Smith has documented the Association's meetings with great precision and has also contributed our first case study. Needless to say, none of this absolves me from total responsibility for the many infelicities which no doubt remain.

LETTERS TO THE EDITOR

Letters should be written with non-APL as well as APL readers in mind wherever possible. 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.

From Mr. Richard Bowman 27th February 1984

Sir: A certain amount of discussion has been going on recently regarding the desirability of an ASCII-character representation for APL. Perhaps I might be allowed to make something of a pre-emptive strike in favour of the present graphic notation.

Many of the present symbols carry something of a visual sense of what they do; for example, ⌽ is concerned with spinning objects around on axes, ⎕ looks somewhat reminiscent of a terminal screen (I don't have to think too deeply about ⎕← sending characters to terminals and ←⎕ taking them from the keyboard), ⍋ is obviously going in the opposite direction to ⍒ .

Leading on from this, APL as presently represented is rather more cross-cultural than the essentially text-represented languages like BASIC (paraphrasing Allen Rose, do French programmers have to write text such as "LAISSEZ A=2 S'IL VOUS PLAIT"?). In its current form I would expect APL to have an advantage in transcending national and cultural barriers.

In any case, while history has imposed a penalty of several hundred pounds as the luxury cost of putting an APL keyboard onto a terminal, we are now in a situation where the STSC (Scientific Time Sharing Corporation) chip for the IBM Personal Computer is being made available for $25. To my naïve mind, this indicates that economics could cease to be the issue.

I suspect that given a historical perspective the advent of APL as a notational style will be seen in the same light as the move from Roman to Arabic numerals, or the evolution of calculus notation. We must not allow ourselves to be tyrannised by the temporary inability or unwillingness of hardware manufacturers to supply us with what we need at an economic price. Some of the introductory remarks in Kenneth Iverson's "Notation as a Tool of Thought" (Communications of the ACM August 1980 Vol. 23 No. 8) are well worth re-reading.

Yours sincerely,

Dick Bowman,
CEGB,
85 Park Street,
London SE1.

(Editor. The various current arguments in favour of promoting a conventional representation for APL do not imply that the existing notation is wholly without any merits, only that its advantages are in some cases outweighed by problems. Dick's argument may well be the right one, but some people feel it is twenty years too late! Most computer users by now already own or have access to non-APL screens; although these could be converted, in practice they won't be. Another issue is the awkwardness of including APL character set output in typeset copy without specialist aids, which most publishers are reluctant to employ. This makes the inclusion of APL algorithms in most computer journals difficult or impossible, limiting in turn the spread of the language. We share this problem; the VECTOR editorial team would dearly like to hear from any organisation that can offer them a solution.)

From Professor A. G. Hawkes 19th March 1984

Sir: I don't know when you expect to bring out the new Journal (to be called "VECTOR") but no doubt you will need a collection of papers to put in it. Enclosed is a paper which I hope you will feel is suitable.

A proper journal should make good sales from libraries. I hope it will cater for a broad spectrum of interests. I think the use of APL for scientific purposes has been sadly neglected in this country, and feel that a proportion of scientific papers in the journal, as well as the more commercial stuff, will help to remedy this.

Yours sincerely,

Alan. G. Hawkes,
Dept. of Management Science & Statistics,
University College of Swansea,
Singleton Park,
Swansea SA2 8PP.

(Editor: We agree with Professor Hawke's views and are grateful to him for contributing a scientific paper, which is included in the Technical Section of this issue).

From Mr. Anthony Camacho 24th February 1984

Sir: I found an error in an algorithm published in the "FinnAPL Idiom List". I think this is rare enough to be worth notice. Idiom 49 does not work if the largest value in "X" is an exact power of 16.

```
     ∇ R←FDTOH X
[1]   ⍝ FINNAPL IDIOM 49
[2]   ⍝ RESULT TOO SHORT IF ⌈/X IS 16*N
[3]    ⍞←'0123456789ABCDEF'[⎕IO+((⌈⌈/16⍟,X)⍴16)⊤X]
     ∇
       FDTOH 16
0
       FDTOH 256
00
       FDTOH 4096
000
       FDTOH 15
F
```

It can easily be corrected by taking the logarithm to the base 16 of "1 + ,X", but if "X" contains many values this process may be time-consuming. It is much faster to find the maximum and to take the logarithm of that value only.

```
     ∇ R←DTOH N
[1]    ⍝ IMPROVED DECIMAL TO HEX CONVERTER
[2]    R←'0123456789ABCDEF'[1+(((⌊1+16⍟⌈/,N)⍴16)⊤N)]
     ∇
       DTOH 15 16
0F
10
       DTOH 4096
1000
       DTOH 4095
FFF
```

This takes less than three seconds to convert the first 256 integers on my microcomputer, compared with about 33 seconds for the improved FinnAPL idiom number 49!

Yours sincerely,

Anthony Camacho,
2 Blenheim Road,
St. Albans,
Herts AL1 4NR.

(Editor: Our Technical Editors confirm Mr. Camacho's observations)

QUICK-REFERENCE APL DIARY

Compiled by David Preedy

Efforts are made to ensure the accuracy of these listings before publication, but readers should in all cases verify details nearer the indicated date. Association members will be posted details of all meetings.

Unless otherwise indicated all Association events are free, open to non-members as well as members, and start at 2 p.m. at:

The Read Lecture Theatre,
Sherfield Building,
Imperial College of Science and Technology,
South Kensington,
London SW7 2BT.
Tel. 01-589 5111
Underground: South Kensington, then subway to Science Museum.

The "APL Publications" bookstall will be a regular feature of all London events, giving delegates a chance to purchase hard-to-find APL literature.

Date	Venue	Event
May 18	London	Association meeting: "APL vs. packages and 4GL's."
June 11-15	Helsinki	APL84 Conference (Details in this issue)
June 22nd	Aldwych	Association Special Event: "APL84 In Focus" (Details in this issue)
September 21	London	Association Meeting: "Expert Systems and APL"
October 15-17	Toronto	I.P. Sharp APL User Meeting: "The Information Centre and Changing Technology" (Details in this issue)
October 19	London	Association Workshop: "How to Survive in Version XX of APL"
November 16	London	Association Meeting: "Information Centre Case Studies"
December	Midlands	Association Special Event: "What's New for APL in 1985?" (Details to be announced)

BRITISH APL ASSOCIATION NEWS

Compiled by Robert Bittlestone

Annual General Meeting

The 1983 Annual General Meeting of the British APL Association (formerly the U.K. APL User Group) took place at 2.00 p.m. on Friday 20th January 1984. Prior to this meeting an unprecedented level of activity had been taking place with a view to sweeping your Association's structure through with a clean broom. The difficulty of organising the Loughborough Business Technology show in September 1983 had put the spotlight on the need for some changes. Now that the dust has settled, we can reveal that the Loughborough event came within a hair's breadth of cancellation three weeks before opening day, primarily owing to a lack of its effective promotion by your Committee! At the last minute we appealed to UK companies trading in APL to help us out, and they generously gave of their time to mail off details to their contact lists. The result was salvation; Loughborough came and went and actually made the Association richer.

"The AGM atmosphere was electric: you could cut it with a knife"

As a result of the Loughborough experience a Task Force was appointed to conduct an in-depth review of all Association activity. The proposals were published in the previous issue of the Association journal and were in essence voted on by the ordinary members at the January Annual General Meeting. It's true that competing candidates were not obliged to swear allegiance to the new manifesto, but social ostracism by the fellow Committee members would have been the inevitable outcome of any officer rejecting the embraces of change. After all, this didn't all happen in 1984 for nothing.

The atmosphere at the AGM was electric; you could also cut it with a knife (an irresponsibly mixed metaphor?). Based on the votes cast there were about 50 enfranchised members present, not all of whom voted on every issue. The first nail-biting scenario occurred when we came to elect a Chairman. Three candidates were presented: Phil Goacher, the outgoing Chairman and a donor of long and worthy service to the Association; Ron Fuss of "Fuss at Loughborough" fame (see last issue) who had been persuaded to stand by a splinter group intent on humour at Committee meetings at all costs; and Dick Bowman who entered the running as a dark horse late in the day, standing on a CEGB platform in apt harmony with the ambience of the meeting.

"The candidates didn't even have $24 million to spend on their campaigns"

After a period of what has been described as "white knuckle time" involving frantic recounts by Anthony Camacho, who had not yet been elected Secretary but was nevertheless constitutional in his arithmetic following Gordon Sutcliffe's request as outgoing secretary for a stand-in, the votes were announced: Bowman 15, Fuss 15, Goacher 16. Neither Hart, Jackson nor Mondale could have played it more fairly, and our candidates didn't even have $24 million to spend on their campaign. The voices of reason held sway and continuity in Association matters was assured, while at the same time our incoming Chairman's majority was such as to leave the balance of power in the hands of the yet-to-be elected Committee.

Worse was yet to come, or better, depending on your electoral viewpoint. Mr. Camacho carried the Secretaryship against Les Hollingbery by 25:15, and Mel Chapman beat Mark Griffiths to the Treasurer's role by 17:12. Mr. Camacho was seen to be putting renewed vigour into his vote-counting now that he really was the Secretary, and Mr. Chapman started to count the cost of proceedings with some financial alacrity.

"The failed candidates' tongues smarting from the bitter taste of defeat"

Next to come was the role of Activities Officer, where the lineup was David Allen from British Aerospace, no stranger to the BCS world with his Kingston branch involvement; Dominic Murphy of Inner Product, no stranger to the APL world with his VIZAPL involvement; Dick Bowman, no stranger to the world of humbler Committee men as distinct from Chairpersons, gritting his teeth with manfully concealed disappointment; and Robert Bittlestone, your humble correspondent, an also-ran for the Journal vacancy and by now eyeing the competition with some concern. Tactical voting prevailed: Mr. Bittlestone withdrew his nomination for the Activities job, posing a new constitutional dilemma for Mr. Camacho by so doing, but our gallant Secretary raised no objection to this irregularity and the Titans were left to fight it out. The result was Allen 14, Murphy 7, and Bowman 17, so Allen and Murphy retired to battle with their tongues smarting from the bitter taste of defeat and Bowman, crowned with a laurel wreath, performed the by now obligatory gladiatorial circuit of the Read Lecture Theatre arena.

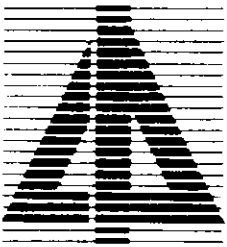
Fred Perkins of I.P. Sharp fame was by now the only candidate for the role of Promotions Officer, and gracefully the ordinary members let him have it. Mr. Perkins obviously felt that an uncontested seat was a worthless seat since he stuck it for three months and then was regrettably obliged to tender his regination (see later). However, we didn't know that at the time and so the auditorium rocked with deafening applause. The post of Journal Editor was then offered and the candidates were Valerie Lusmore, the outgoing Editor, an APL person from APL People and the worthiest possible candidate following her years of dedication to the cause, and Robert Bittlestone of MicroAPL and Metapraxis, whose only claim to APL journalistic fame was his previous editorship of the blatantly commercial "MicroAPL News". Would integrity prevail over commerce? Very nearly, but after a further ashen-faced recount Mammon triumphed by 21 votes to 20, ushering in by one head the current regime of logorrhoea.

"A wave of almost invisible releif swept over the AGM audience"

After this almost unbelievably stressful sequence it was with some visible lack of tension that Chris Beatty of Westminster City School was elected unopposed as Education Officer, and that Romilly Cocking of Cocking & Drury was elected also unopposed as Technical Officer. A wave of almost invisible relief swept over the audience and voting finished with an almost unnoticed ruse of great cunning by the new Chairman, to extend all our terms by 3 further contest-free months: the Association current year-end has now been fixed at 30th April 1985 and annually thereafter, thus conveying yet headier joys of this extended office on our unworthy heads.

"The acronym formed by 'British APL Group' was rejected"

Talking of "Association" reminds me to announce that the name of your Organisation has undergone various metamorphoses. The original name of "APL User Group" was felt to be too low-key to live up to the major expectations that are now had of it, and so "British APL Society" was proposed. However, the British Computer Society were unenthusiastic about this since it might imply that all British Computers were in fact running APL, a commendable notion but rather short of the facts. So "British APL Group" was coined and lived for about two months until your Editor discovered to his horror the appalling nature of the acronym thus formed. Nights were lost with visions of seething masses of APL enthusiasts exclaiming "It's in the BAG", so the BCS was persuaded to adopt BAA instead. Any resemblance to a national airport authority is felt unlikely to pose a major practical problem. We hope our readers approve of this nomenclature.



COCKING & DRURY LTD.

THE APL PROFESSIONALS

If you are an experienced APL user, you will know how much can be achieved with APL - and will recognise the skills needed to use the language to its full potential.

If you're new to APL, or are evaluating the language, you will want professional and unbiased advice and assistance.

We have over 100 man-years' experience of APL, and we are familiar with all major mainframe, mini and micro implementations of the language.

Cocking & Drury were the first European APL consultancy, and we are still the leaders in

applications programming

education

packages

turnkey systems

conversion

consultancy

information centre implementation

systems programming and support

For further information, contact Romilly Cocking on

01-493 6172

or write to us at

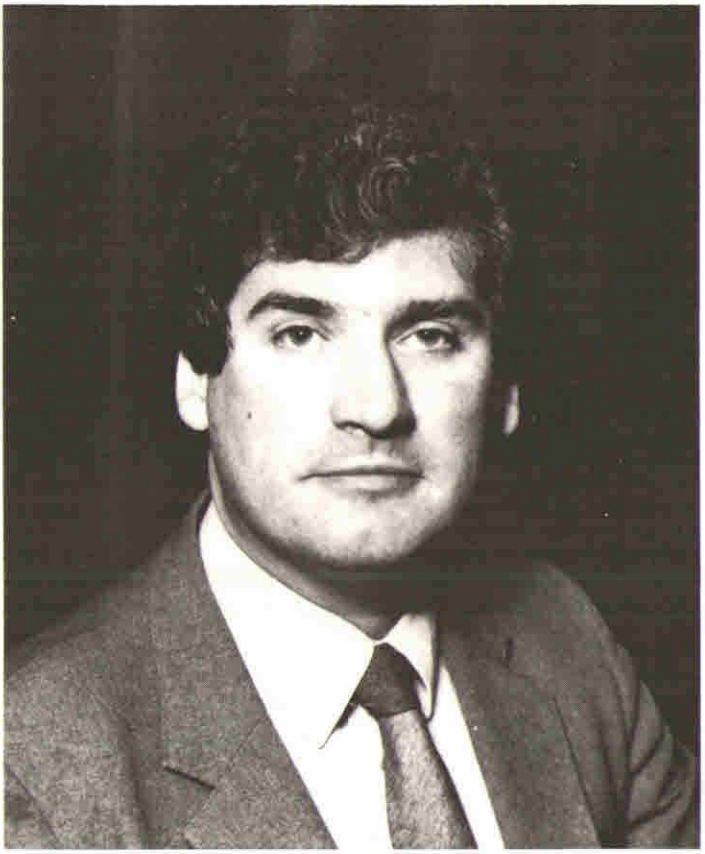
16 Berkeley Street, London W1X 5AE

APL84 in focus

Many UK APL enthusiasts will be unable to visit the APL84 conference in Helsinki this year. Dick Bowman, the Association's Activities Officer, has thoughtfully provided a special session whereby speakers travelling back from Helsinki via London will be waylaid and asked to regurgitate their seminars. After the practice in Finland the presentations should be even sharper, so don't miss this opportunity of hearing the highlights of APL84.

This is a Special Event of the Association and is being held at the Waldorf Hotel, Aldwych, London WC2 on Friday 22nd June. A charge will be made, which will include lunch. The speakers will include Jim Brown of IBM, who will be talking about APL2; Roy Sykes of STSC, the author of "Collected Whizzbangs"; Philip van Cleave of TCC and MicroAPL talking about APL.68000; and a demonstration of the Analogic Corporation "APL Machine".

For technical details contact Dick Bowman at the address at the end of the journal. For booking inquiries contact Philip Goacher at the BCS, address also at the end of the journal.



Our thanks are due to Romilly Cocking for sterling services to the APL User Group in the past and to the formation of the current Association

Situations Vacant

Mr. Romilly Cocking has felt obliged to tender his resignation as Technical Officer. Mr. Frederick Perkins has also felt obliged to tender his resignation as Publicity Officer. Both these posts are currently vacant and applications are invited to fill them.

The role of Technical Officer requires a history of exposure to APL, ideally on several different types of computer, and an acknowledged expertise in the technical aspects of the language. Duties include (a) acting as a reference point for members of the Association who have technical inquiries, (b) coordinating the activities of the Association in relation to APL standards, (c) supporting the Journal Editor by identifying potential authors, and (d) supporting the Activities Officer by identifying speakers and panellists.

The post of Publicity Officer requires an ability to handle Press contacts and a gift for phrasing announcements about Association affairs in an interesting and news-worthy way. Duties include (a) handling all Press matters, including press releases (b) liaising with sustaining members over matters of general APL market interest, (c) assisting the Journal Editor in promoting the circulation of VECTOR, and (d) helping the Activities Officer to publicise Association meetings.

The Association statutes are such that no general vote of members is required to fill interim vacancies. Consequently these officers will be appointed by vote of the Committee after reasonable time has elapsed for candidates to make their interests known. Please write to the Secretary of the Association (address at the end of this issue) with a brief outline of your APL experiences, a statement of your interests and orientation if appointed, and a recent black and white photograph.

APL86

International APL Conferences take place alternately in the USA and elsewhere. The 1982 Conference was in Heidelberg; the 1983 in Washington; 1984 takes place in Helsinki; 1985 in Seattle, and 1986 has been allotted by EuroAPL to the United Kingdom. Initial planning is currently under way by your Committee, and in due course it is likely that a special secretariat will emerge to run the affair, with an identity distinct from the Association Committee itself, although existing officers are by no means disbarred.

Offers of assistance in organising APL86 will be gratefully received by the Secretary. There is also a requirement for referees to scan through papers that are submitted to the conference. Although such appointments are normally by invitation, any individual who considers himself or herself qualified and enthusiastic for such involvement is also invited to contact the Secretary.

1983 REPORT AND ACCOUNTS

Jonathan Barman

INCOME AND EXPENDITURE FOR YEAR TO:	31.10.83	31.10.82
	£	£
Income		
Subscriptions	1,392	1,190
APL Conference Nov. 82	1,183	
APL Conference Sep. 83	5,193	(149)
Profit on sale of books	33	291
Total Income	7,801	1,332
Expenditure		
Meetings	334	384
Newsletter less advertisement income	(95)	407
Expenses	169	260
Audit fee	60	30
Total expenditure	468	1,081
Surplus of income over expenditure	7,333	251
BALANCE SHEET AT YEAR END		
Cash at bank	3,331	322
Debtors	5,890	694
Books in stock	0	650
	9,221	1,666
Less Creditors	(706)	(484)
Net Assets	8,515	1,182
Accumulated surplus previous year	1,182	931
Surplus current year	7,333	251
Net Reserves	8,515	1,182

NOTES TO THE 1983 ACCOUNTS

1. The surplus on the 1983 APL Business Technology conference was made up as follows:

	£
Delegates fees	17,340
Exhibitors' fees	3,555
Programme advertising	375
Product forum fees	300
Total Income	21,570
Conference costs	13,662
Publicity costs	2,031
Miscellaneous costs	684
Total costs	16,377
Surplus of income over costs	5,193

2. Pending expenditure of the accumulated surplus, the cash is being invested to produce further income.

3. On 1st November 1982 the book stock was sold to APL Publications, run by Mr. L. Hollingbery. The committee decided that the administration of book sales was becoming too onerous for it to be carried out by members of the committee, who receive no fees for work carried out on behalf of the User Group.

(The accounts have been signed by Jonathan Barman as Treasurer and audited by Anne Raikes MA FCA).

19th January 1984

Editor's comment: The Loughborough conference swelled the finance of the Association substantially and is in part the reason for the expanded programme of activities and journal content that your Committee has embarked on for the 1984/85 year.

INTERNATIONAL APL NEWS

Compiled by Robert Bittlestone

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. A substantial number of VECTOR subscriptions are from overseas and we hope in this way to keep APL users in touch with each other internationally.

EuroAPL

EuroAPL is the international organisation supporting European APL groups in various ways. We had hoped for a glowing progress report on news of APL in Europe, but sadly this is not to be — not in this issue anyway. Philip Goacher is this year's Chairman of EuroAPL as well as being the Chairman of our Association. He writes somewhat ruefully:

> "EuroAPL has been supported from funds allocated by the Commission of the European Communities to the 'Multi-Annual Programme for Data Processing', which has allowed us to meet the travelling costs of members attending the regular meetings in Brussels."
>
> "Unfortunately the proposal for extending these funds was not approved in March and the budget has now run out!"
>
> "Whilst waiting for Council to reach a decision on continued finance for the work carried out under the programme — principally standardisation work — all meetings for the various activities, including EuroAPL, have ceased."
>
> "Members of the EuroAPL Board attending APL84 will meet to discuss the EuroAPL programme during the conference."

So much for entente cordiale: it's fine so long as someone pays for it. We hope to bring you more stimulating news of EuroAPL in the next issue.

I.P. Sharp 1984 APL User Meeting

For those of you with the opportunity (or excuse) to visit Canada, the regular Sharp APL user meeting is being held at the Westin Hotel, Toronto, from October 15th — 17th 1984. The following details are reprinted from a public mailbox message on the IPSA system.

> "The Information Centre and Changing Technologies"
>
> "Featuring keynote addresses by H. Mitchell Watson Jr., IBM Vice President and President, System Products Division, IBM Corporation, White Plains, New York; and by James Cunnie, Director of Office System Planning, ITT Corporation, Stratford, Connecticut. There will be a special half day presentation by James Martin, who is according to Computer Weekly the computer industry's most widely read author and best attended lecturer."
>
> "Session themes include:-
>
> — Introducing the global information centre
> — Implementing a global information centre
> — Menus, mice and masters in application design
> — Managing the information revolution
> — Evaluating and implementing user productivity tools
> — Managing and implementing international systems
> — Some management tools for an information centre

— Supporting information centre users
— Exploiting networks
— I. P. Sharp: changing technologies and future directions
— The evolution of the APL environment: two perspectives

"Optional tutorials (October 18th) cover:-

— For managers The Sharp APL system: a management perspective
— For technicians Dump analysis facility

"The registration fee for the 1984 meeting is $400 (Canadian) or $320 (US). There is an additional fee of $100 (Canadian) or $80 (US) to register for either one of the tutorials. A conference brochure and registration form will be mailed in early May to everyone on the IPSA newsletter mailing list. For more information contact:

Rosanne Wild, IPSA Ltd., Suite 1900, 2 First Canadian Place, Toronto, Ontario, Canada M5X 1E3. IPSA Mailbox RWI. Telephone (416)-364-5361

The feedback from previous Sharp conferences has always been very good, with an accent on the practical as opposed to the theoretical. It is not normally necessary to be an IPSA customer to be allowed to register and many users of IBM APL systems or of microcomputer-based APL have learned a lot from the Sharp conferences.

Having bestowed praise where it is due, perhaps I might raise a pet complaint. Users of timesharing bureaux, and particularly their staff, seem to imagine that the APL mailbox is the only possible or civilised way of sending someone a message or distributing news. I couldn't disagree more. For it to be practical use, one has to inspect a personal mailbox at least daily, and I certainly have neither the time nor inclination for that. A telex or a letter is actually delivered personally, a great innovation.

What's more, if I want to send someone a message on a mailbox system, in practice I have to type it personally as opposed to being able to delegate it to a secretary (in theory the secretary could log on for me but in practice the lines are always down at the crucial moment). It also arrives the other end on some kind of junk printer: as sender I have no control over a tidy format. Finally, there's no lower case, which makes all my messages look like imperatives! Mailboxes may be fine for programmers but as human beings, please let's be civilised (and I haven't even mentioned the cost!).

APL84

By the time this issue of VECTOR is published, you may or may not have enough time to book for the APL84 International Conference at Espoo, near Helsinki, Finland, which starts on 11th and ends on 15th June 1984. A most splendid programme has been prepared, which is unfortunately too long for us to reprint here. What you will need for a late booking is a few addresses. The company handling all reservations for the conference and accomodation is:

AREA Travel Agency Ltd.,
Congress Department,
P.O. Box 227,
SF-00131 Helsinki 13.
Telex 122783 ARCON SF
Tel. Int. + 358-0-18551

There is a full range of tours available, including one to Leningrad for which a visa is required.

Late bookers in the UK can contact the London-based company which is handling group bookings to see if their April deadline can be varied. The details are:

Noel Cramer,
Express Boyd Ltd.,
4/5 Bonhill Street,
London EC2A 4BX.
Tel. 01-588-8461

At the time of preparing for publication the package price for travel and accommodation varied from £328 to £498 per person depending on hotel and timing. Added to this is the cost of conference registration itself, which starts at £245 and increases asymptotically to infinity, depending on your predeliction for tours before or after the conference. All in all an expensive conference but without doubt a very worth while one — and a rare chance to visit Finland or the USSR on company business!

A Finnish Fairy Tale

No doubt the APL84 conference organisers have been very busy with preparing for the show. I started to get a little concerned about communications with Helsinki back in November 1983, when I wrote to Sten Kallin, who is one of the organisers, to ask if they would like to run the "Metaconference" as a service to delegates. (This is a form of computer-based interaction system that lets users contribute statements and also vote on everyone else's; interested readers can catch an account of it in the Journal of the UK Operational Research Society December 1983.)

I waited till late January 1984. No response.

I wrote again. No response.

In desperation I sent Sten Kallin a telex message care of AREA travel agency, addressed via Timo Seppala, the conference chairman. It said:

> "I have never received any reply to my letters to Sten Kallin of 11th November 1983, 26th January 1984 and 7th March 1984. I am going to spread a rumour that he does not exist. Is this true? Please let me know."

No response.

Now I do know that Timo exists — at least he did in Washington, and his voice on the telephone last year sounded the same as it did when I met him in the flesh. But what about Sten Kallin? Is he in fact a virtual person, overlaid on some disc somewhere underneath a head crash? Is his name some kind of anagram? APL is actually quite good at generating anagrams. You type in the text STEN KALLIN, assign it to a variable called X or whatever, and then watch APL executing X[11?11] in a tight loop. You get the occasional duplication that way of course, but it's nothing to lose sleep about. Actually you might lose a little sleep come to think about it because the effect on the screen is most hypnotic. Try it. I bet you won't be able to hit the break key until at least one credible anagram emerges.

I started by getting the head-scratching NIT KALLSEN, then the well-known Baltic sailor SALT ENKLIN, and shortly after that the legendary giraffe impersonator IN TALLNEKS. You can also very nearly get LINK NATES, which is really quite worrying if you happen to know one of the two meanings of that obscure English word (I don't mean "anterior pair of optic lobes in brain", although that would probably be confusing as well). The mysterious electrical researcher TESLA is buried in there, as is some EKTASI. But I felt I had hit on a clue when up jumped the phrase

KATE LSI NNL. Sten Kallin has had a sex change and subsequently been blown into LSI logic (large scale integration). I think the NNL is the last human gasp before digitisation, but I can't be sure.

End of fairy tale — or should I say finish. Access to the pages of VECTOR is a privilege which I have undoubtedly abused by now. It's really much safer to answer people's letters! I'm sure the conference will be very well organised, the more so for people like Sten not wasting their time writing to me. I see from the programme that Espoo, the conference venue, is an archaeological site dating back to 6000 B.C., so no doubt there will be a crop of jokes about digging up APL hieroglyphics. You may also catch a rare chance to sample the little-known Finnish custom of culinary monuments. To quote (just about) from the programme:

> "Espoo is now the fourth largest city in Finland, featuring the Garden City of Tapioca as well as other achievements of modern Finnish architecture".

The more hasta, the less pasta? Have a good trip!

BRITISH COMPUTER SOCIETY NEWS

Compiled by Philip Goacher

I suspect that few of our members know much about the activities of the British Computer Society (BCS) of which we are one of the 44 Specialist Groups. Indeed, despite being a BCS member for ten years and as APL Group Chairman sitting on its Specialist Group Board, it was not until I became the BCS Business Manager recently (Ed. — see Press release after this article) that I came to understand more fully the workings of the Society.

The BCS was formed in 1957 as a result of meetings between two informal groups, one representing industrial and commercial users of computers, the other representing scientists and engineers who were using computers in their disciplines. After its first year of existence the Society had 1,182 members; it now has a membership of 27,500 which represents a considerable proportion of practising computer professionals.

The cornerstone of the Society's existence is its representation of the computing profession in the UK, and the status of Professional Membership of the BCS is recognised throughout the industry. Professional members are required to subscribe to a Code of Conduct concerned with integrity, confidentiality and social responsibility. A Code of Practice is provided for the guidance of members and is designed to promote the efficient and responsible use of computers.

"Since 1957 the Society's membership has grown from 1,182 to 27,500"

The result of this recognition of the status of the Society is that it is the accepted mentional voice on such professional and social issues as standards in education, good practice and data protection. The Society also has international links with many computer organisations, particularly the International Federation for Information Processing (IFIP) and representative bodies of other countries such as ACM (USA), ACS (Australia), CIPS (Canada), CSI (India), AICA (Italy), DARA (W. Germany) and AFCET (France). The Society also develops its ability to represent wider areas of the profession through its boards, committees, specialist groups and branches.

Another important function of the Society is that it should inform and educate in all areas of computing. It does this through its quarterly publications "The Computer Journal" and "Computer Bulletin", its page of news and events in "Computing", as well as via technical monographs, reports and conference papers. The BCS holds conferences, lectures and seminars in all aspects of computing, often organised by its specialist groups. Each year a number of lectures is organised by the Society; recently these have featured Grace Hopper, Gordon Bell and Gene Amdahl. There is also an Awards scheme in which prizes are given to the most deserving entrants in each of three categories: technical, social benefit and applications. The purpose of these awards is to promote and publicise achievements in the computing industry.

"Recent speakers include Grace Hopper, Gordon Bell and Gene Amdahl"

This process continues at a local and specialised level via the Society's 42 regional branches and 44 specialist groups. The Society also maintains the largest computing library in the UK at the IEE; this is open to personal callers and also offers members a postal lending service.

Membership of the Society is open to any person involved either directly or indirectly in computing. The requirements of each class are under continuous review to allow the Society to respond to new situations. The professional grades are Associate Member (AMBCS), Member (MBCS) and Fellow (FBCS), advancement depending on level of professional education and evidence of experience of computing over a specified number of years.

There are also the grades of Affiliate, Graduate and Student. Affiliates are those interested in computing who wish to become involved in Society affairs. Graduates are those who have satisfied the Society's educational requirement but lack the experience necessary for professional membership, while Students are those following a course of instruction recognised by the Society. Members of these grades enjoy most of the same privileges as those in the professional grades except that they have no voting rights.

The Society is currently applying for a Royal Charter, which would lend additional weight to the Society's status as the computing profession's representative organisation.

If you are interested in applying for membership of finding out more about the BCS, please write to me at the address on the inside back cover of this issue. Through this column I will keep APL Association members informed of BCS developments which affect our Specialist Group.

BCS Appoints a Business Manager

Editor — I am sure we would all like to congratulate Philip Goacher on his recent appointment as Business Manager of the British Computer Society. Here is an extract from the recent Press Release concerning his new role, omitting only Philip's biographical details to spare his blushes:

> "Philip Goacher joined the BCS as Business Manager in January this year. His brief is to develop further the Society's publications and to provide a range of services including support to the Society's Specialist Groups and Branches in their aim to widen their sphere of influence, particularly through conferences."
>
> "From 1st May 1984 these activities are being managed by The British Informatics Society Limited (BISL), the commercial subsidiary of the BCS, with Philip Goacher as Managing Director."
>
> "The BCS makes contact, not only with its members, but also with the world at large, through its various publications and reports, and through the many conferences of high technical quality which are held regularly. By developing the professional support given to these and other growing activities, BISL will be helping to promote the role of the BCS within the computing profession."
>
> "Through his interest in APL, Philip Goacher has been closely involved with the BCS for some time as Chairman of their APL Specialist Group. He is also Chairman of EuroAPL, an international liaison group supported by the EEC."

NEWS FROM SUSTAINING MEMBERS

Compiled by David Preedy

The category of "Sustaining Membership" has been introduced and is being 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 this section of the journal, together with opportunities 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*Plus Ltd.	Aston Science Park, Love Lane, Birmingham B7 4BJ. Tel. 021-359 5096
Cocking & Drury Ltd.	16 Berkeley Street, London W1X 5AE. Tel. 01-493 6172
Dyadic Systems Ltd.	30 Camp Road, Farnborough, Hants. Te. 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	132 Buckingham Palace Road, London SW1W 9SA. Tel. 01-730 4567

Despite being offered free publicity in this section of VECTOR, only two of these companies provided us with news in time for the print deadline! We're not sure whether the reason was modesty or inefficiency but no doubt we shall be inundated with copy for the next issue.

APL*PLUS Limited

"APL*PLUS's parent company STSC Inc. has won two I.P.C. awards, recently presented in a joint ceremony in London and Dallas."

"APL*PLUS/PC won a 'Million-in-One' award for achieving $1,000,000 of business worldwide in its first year, and "APL*PLUS 1000 won a 'Million-Dollar-Club' award for reaching sales of $1,000,000 since its introduction."

"Since the New Year, "APL*PLUS has announced three new products."

"A range of Information Centre tools provides full-screen data entry, display and analysis capabilities, and interfaces to other proprietary Information Centre products."

"Release 3.0 of "APL*PLUS/PC now offers a colour graphics system in addition to their

extended APL for the IBM PC and compatibles."

"Finally, a financial and statistical package has been released containing more than 200 programs for the IBM PC."

MicroAPL Limited

"MicroAPL has extended its product range in both larger and smaller systems."

"The SPECTRUM supermicro now offers up to 10 Mb of main memory and supports 144 Mb hard discs. Customers with larger systems can now use the Tektronix 4105 colour graphics terminal to which MicroAPL has added an APL character set. They are currently testing software to emulate IBM's GDDM colour graphics processor."

"At the smaller end of the product range, MicroPLOT, which offers business graphics to drive the IBM PC's colour monitor and Hewlett-Packard plotters, is now available under "APL*PLUS/PC, and the MicroSPAN self-teaching package will soon follow."

"On the personnel front there are three new faces at MicroAPL. Emma Saunders has joined as personal assistant to three of the directors; Karen Salt has become personal assistant to the Managing Director, Richard Nabavi; and Alastair Kinloch, well-known in the Scottish APL community, has joined the team of applications consultants."

APL PRODUCT GUIDE

Compiled by David Preedy

Here is the first release of our exclusive new APL Product Guide, which we hope will provide VECTOR readers with useful information about sources of APL hardware and software. We shall be updating this with each issue of VECTOR and we depend on the alacrity of suppliers to keep us informed of their products.

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 & XT	1995 – 4425	Authorised IBM PC dealer supplying complete systems including peripherals based on IBM PC.
Cocking/Drury	MicroAPL SPECTRUM SAGE II SAGE IV	6000 – 35000	Supplied as part of a turnkey system. For product details see MicroAPL entry.
Inner Product	IBM PC	2000 – 5000	All IBM PC compatible software available.
MicroAPL	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 – 600	See APL*PLUS entry
APL*PLUS	APL*PLUS/PC	600	Full feature interpreter for IBM PC and PC/XT. Also for compatibles CORONA, COMPAQ and COLUMBIA.
	APL*PLUS/1000	27000 – 47000	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.

Company	Product	Prices £	Details
APL INTERPRETERS (continued)			
Inner Product	VIZ::APL	255 – 360	8-bit Zilog Z-80 CP/M
	APL*PLUS/PC	600	See APL*PLUS entry
I.P. Sharp	Sharp APL	poa	For IBM mainframes
	Sharp APL/PC	poa	For IBM PC
MicroAPL	APL.68000	1000 +	Full implementation with component files, error trapping etc. for SPECTRUM and SAGE machines or other popular MC68000-based computers.
APL VISUAL DISPLAY UNITS			
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.
Lynwood	Alpha	1995	Full APL character set, 16-bit microprocessor with pixel addressable monochrome graphics.
	Alpha Colour	2985	As above but 8 colour graphics.
MicroAPL	Insight VDT-1	795	Inexpensive APL VDU
	Insight GDT-1	1450	With monochrome graphics
	Tektronix 4105	2995	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, two comms. ports, low profile keyboard.
	Concept GVT/APL +	1370 – 1825	As above with vector graphics.

Company	Product	Prices £	Details
APL PRINTERS			
Alan Pearman	Epson FX series		Printer prices range from 289 to 1995 over all models.
	NEC Spinwriter		
			PRISM 80 PRISM 132
Datatrade	Datasouth DS180	1395	180 cps matrix printer with 2000 byte buffer, 9x7 dot matrix and APL option.
Inner Product	Epson FX80	500	Soft char. set, 160 cps, 80 col.
	Anadex 9620	1150	200 cps., 132 col., tractor feed
	Siemens PT88	620	180 cps., 80 col., silent
	TGC Starwriter	1180	40 cps., letter quality
MicroAPL	Datasouth DS180	1545	See Datatrade entry
	Philips GP300	2251	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	poa	Comprehensive package for IBM PC and compatibles.
Cocking/Drury	ARMS 2	3000 – 15000	Applications Generator
	AFM	10000	High performance shared file system.
	CALL/AP	3000	Non-APL program execution.
	FORMAT	2250	Enhanced report formatting
	WSAIDS	795	Workspace documentation and development aids.

APL PRODUCT GUIDE (continued)

Company	Product	Prices £	Details
APL PACKAGES (continued)			
Holtech	CASH	3500 – 10000	Accounting package and hotel management system implemented on MicroAPL SPECTRUM & SAGE CPUs.
Inner Product	Viewcom	150	Control Viewdata from APL
	APL/DBASE II	150	Interace APL with dBase II
	APL/WORDSTAR	150	Interface APL with Wordstar
	APL/MULTIPLAN	150	Interface APL with spreadsheet
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
	MAGICSTONE	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
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

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
	Employment Agency	poa	Permanent employees placed at all levels. Contractors supplied for short or long term projects, supervised.
Cocking/Drury	Consultancy	120-150 140-200 185-300 275-400	Junior consultant Consultant Senior consultant 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.
Inner Product	Consultancy	200	Communications a speciality
I.P. Sharp	Consultancy	poa	Consultancy and support service world-wide.
MicroAPL	Consultancy	150 – 250	Technical & applications consultancy.
OTHER PRODUCTS			
I.P. Sharp	Productivity Tools	poa	System utilities, operations utilities, system administration utilities, analyst utilities, auxillary processors, comms. software, int. comms. network.
	Databases	poa	Financial, aviation, energy and socioeconomic.

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. 0223-207530
APL*PLUS Ltd.	Barrie Webster John Ward	Aston Science Park, Love Lane, Birmingham B7 4BJ, W. Midlands. 021-359 5096
Burroughs Machines Ltd.	M.J. Fennell	Heathrow House, Bath Road, Hounslow, Middlesex TW5 9QL 01-750 1400
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
Farnell International Instruments Ltd.	Paul Ferguson	Jubilee House, Sandbeck Way, Wetherby, W. Yorks. 0937-61961
	or Roger Attard	Davenport House, Bowers Way, Harpenden, Herts. 05827-69071
Holtech Ltd.	Jan Bateman	44 Conduit Street, London W1R 9FB. 01-734 7618
Inner Product Ltd.	James Manning	Eagle House, 73 Clapham Common Southside, London SW4 9DG. 01-673 3354
I.P. Sharp Associates Ltd.	David Weatherby	132 Buckingham Palace Road, London SW1W 9SA. 01-730 4567
Lynwood Scientific Developments Ltd.	Gareth Wokes	Park House, The High Street, Alton, Hants GU34 1EN. 0420-87024

VENDOR ADDRESSES (continued)

Company	Contact	Address & Telephone
MicroAPL Ltd.	Richard Nabavi	Unit 1F, Nine Elms Industrial Estate, 87 Kirtling Street, London SW8 5BP. 01-622 0395
Shandell Systems Ltd.	Maurice Shanahan	12 High Street; Chalfont St. Giles, Bucks HP8 4QA. 02407-2027

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.

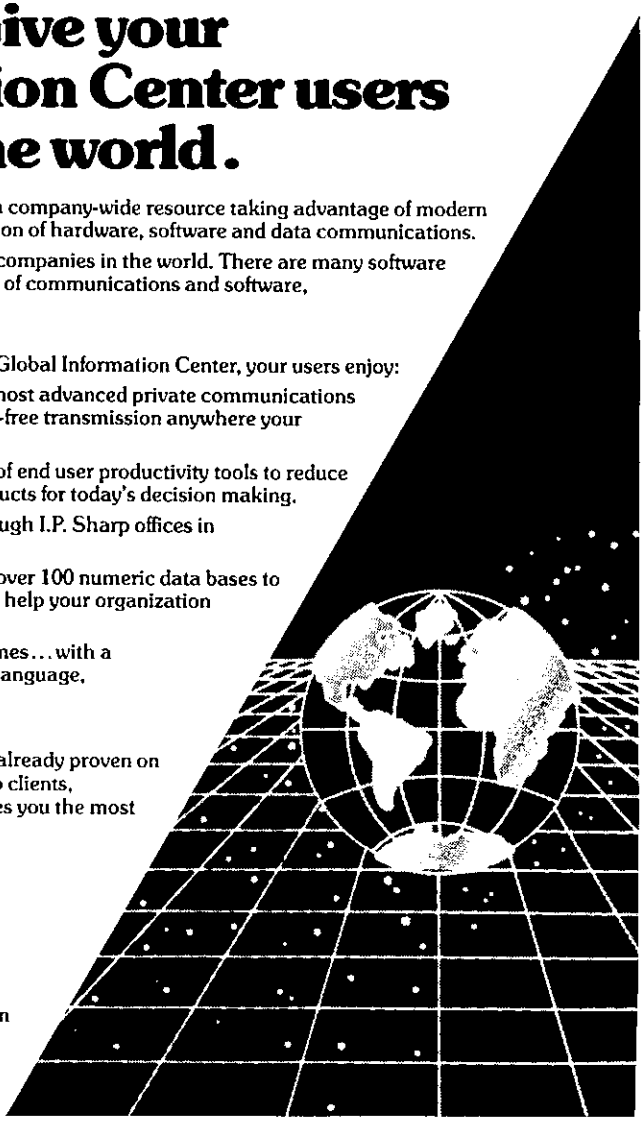
Give your Information Center users the world.

An Information Center should be a company-wide resource taking advantage of modern technologies through the integration of hardware, software and data communications.

There are many communications companies in the world. There are many software companies. But for full integration of communications and software, there is only one I.P. Sharp.

With I.P. Sharp's products for the Global Information Center, your users enjoy:

- IPSANET... one of the world's most advanced private communications networks providing secure error-free transmission anywhere your company operates.
- Integrated Software... a library of end user productivity tools to reduce your applications backlog. Products for today's decision making.
- Worldwide User Support... through I.P. Sharp offices in 22 countries.
- Online Public Data... access to over 100 numeric data bases to complement corporate data and help your organization make informed decisions.
- Integration of PC's and mainframes... with a single application development language, SHARP APL.



Ready to install, ready to run, and already proven on an international scale by I.P. Sharp clients, the Global Information Center gives you the most advanced implementation of the Information Center concept.

Why not let them have the world?

Call your nearest I.P. Sharp office in one of the cities listed below and we'll show you how your Information Center users *can* have the world.

I.P. Sharp Associates

Products for the Global Information Center

London, England (European Headquarters)
I.P. Sharp Associates Limited,
132 Buckingham Palace Road, London SW1W 9SA,
England. (01) 730-4567. Telex: 8954178 SHARP

Aberdeen, Amsterdam, Atlanta, Boston, Brisbane, Brussels, Calgary, Canberra, Chicago, Copenhagen, Coventry, Dallas, Denver, Dublin, Dusseldorf, Edmonton, Frankfurt, Halifax, Helsinki, Hong Kong, Houston, London, Los Angeles, Madrid, Melbourne, Mexico City, Miami, Milan, Montreal, New York City, Newport Beach, Oslo, Ottawa, Palo Alto, Paris, Rochester, Rome, San Francisco, Saskatoon, Seattle, Seoul, Singapore, Stockholm, Sydney, Tokyo, Toronto, Vancouver, Victoria, Vienna, Warrington, Washington, Wayne, White Plains, Winnipeg, Zurich

APL AND GRAPHICS

Compiled by Adrian Smith

The Association meeting on Friday 10th January was well-attended, bearing in mind the normal lack of enthusiasm for an Annual General Meeting, and was marked by the announcement of an Acornsoft APL for the BBC Micro. The official business of the day was followed by three brief talks on the general theme of APL and graphics.

Chris Lee of APL*Plus introduced Release 3.0 of APL*Plus/PC, which runs with several different graphics cards, including the IBM colour card and the Hercules monochrome card, to give business or free-form graphics. It achieves this with a veritable shoal of quad-functions — for more details see the APL*Plus handouts.

Graeme Robertson of I.P.Sharp Associates followed with a slide show of "A Graphic Vision of APL Data and Functions". He has provided us with some examples of his slides in the next article.

The meeting ended with Mr. Martin Martin demonstrating an example of an APL system which employed the IBM 3270 Graphics attachment to drive a circuit-design application.

A GRAPHIC VISION OF APL

by Graeme Robertson

Many graphics packages written in APL have appeared on the I.P. Sharp Associates system over the last seven years. Their capabilities are indicated in some forty slides prepared on a flatbed plotter using software packages SUPERPLOT, SAGA, DRAWMAPS, ROUTEMAPS, STARMAP, DRAWNET, PLOT3DH, PRISMPLOT, PICTURES and GRAPHICS.

A close analogy exists between the vector algebraic approach of APL notation and that of GRAPHICS, the inspiration behind these other packages.

GRAPHICS is used to represent APL arrays of different ranks. Lines are generated which represent real and complex numeric vectors. Surfaces are generated which represent real numeric matrices. Solid wireframe lattices are used in the representation of three and four dimensional arrays. Finally, all these attempts are abandoned in favour of a recursive definition which can in principle yield a visual representation of a real or complex numeric APL array of any rank whatever.

These results were exhibited in the seminar using twenty colourful slides. Arrays which are generated as the results of APL functions give very pleasing images which, it was argued, can be used to help students to understand the nature of complicated mathematical functions.

Structure of the Seminar

1. An Overview of Computer Graphics

    1.1 A definition of Graphics
    1.2 A cursory history of computer graphics
    1.3 Application areas

2. From Business Graphics to Art

    2.1 Towards Graphical Representations
    2.2 Business Graphics
    2.3 Into 3-D
    2.4 Art

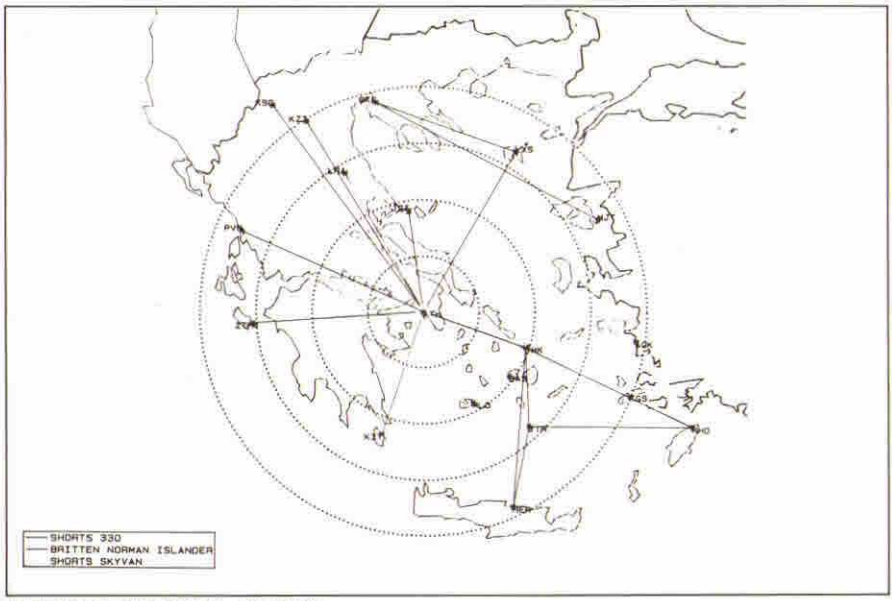
3. APL and Graphics

    3.1 Graphic Variables
    3.2 Graphic Primitive Functions
    3.3 A Graphic Operator
    3.4 Graphic APL-like Utility Functions

4. Representations of Numeric Data and Functions

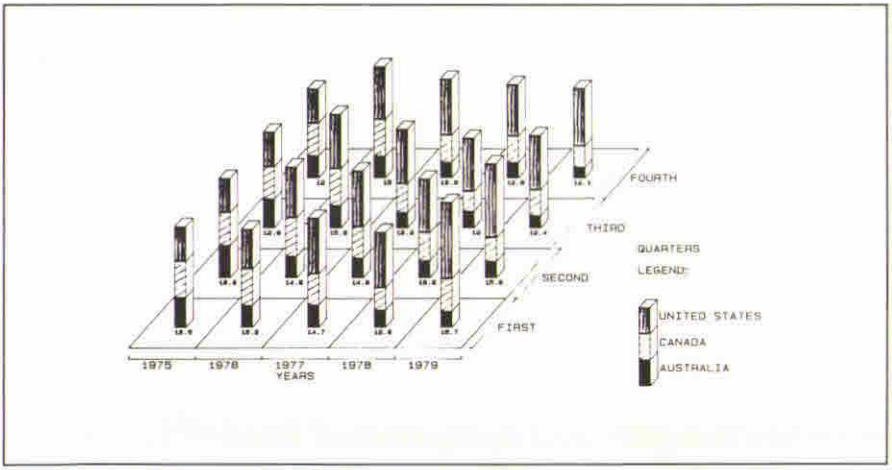
    4.1 Result Rank 1
    4.2 Result Rank 2
    4.3 Result Rank 3
    4.4 Result Rank 4
    4.5 Arbitrary Rank

Fig.1 — Greek Domestic Light Transport Flights for May 1983



COURTESY OF I.P. SHARP ASSOCIATES. (01) 730 4582.
OAG DATA COPYRIGHT 1984, BY OFFICIAL AIRLINE GUIDES INC., OAK BROOK, ILLINOIS

Fig.2 — Total Reserves minus Gold (Billions of U.S. Dollars)



PRODUCED USING <PLOT3DH>.....I.P. SHARP ASSOCIATES

Fig.3 — I.P. Sharp Associates London Office Location

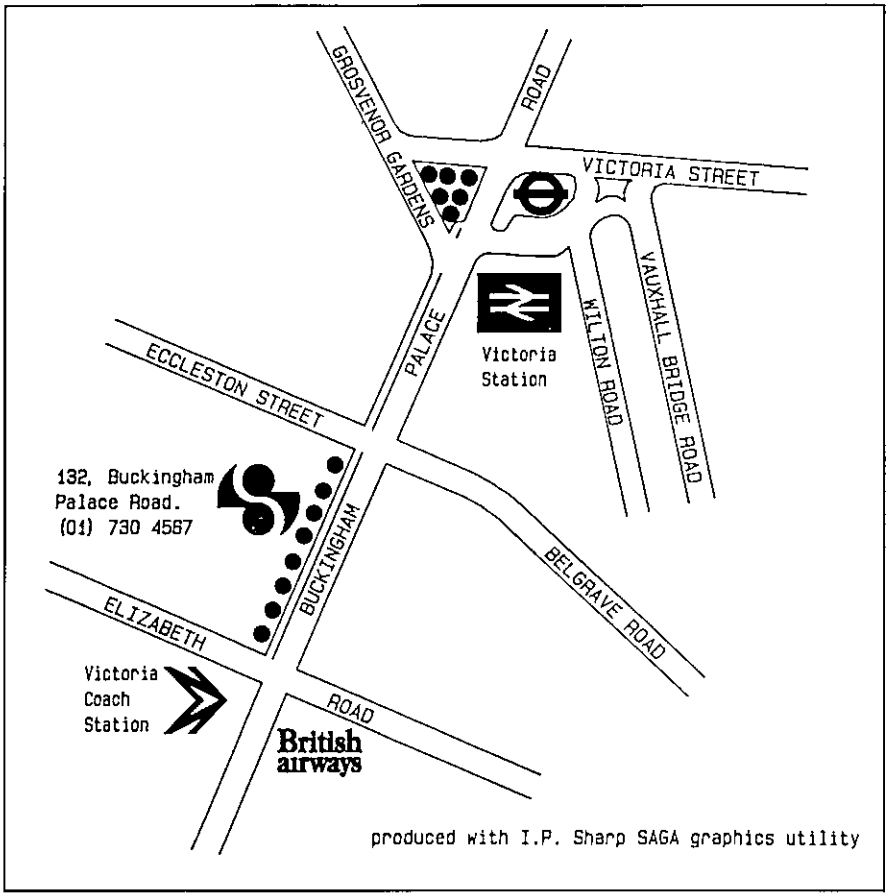
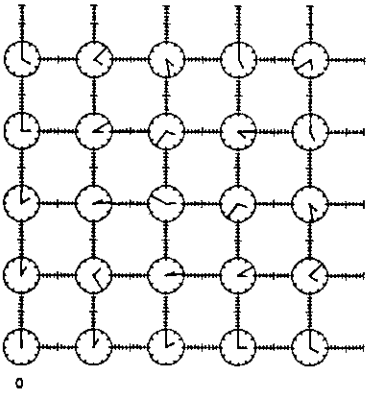


Fig.4 — Computer Aided Instructions Using Graphics



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 his. 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 A and two in B. One in A and one in B are coincident when the light departs, and one in A and one in B 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 A reads AT and the second clock in B reads BT at a place and time the light arrives. According to A, the B clock has moved a horizontal distance

$$ABX1VxAT$$

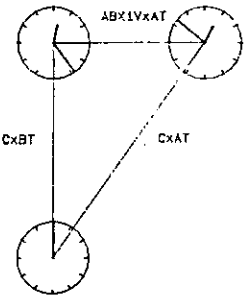
and the light has travelled a diagonal distance

$$CxAT$$

while for B the light has travelled a vertical distance

$$CxBT$$

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 A and B as

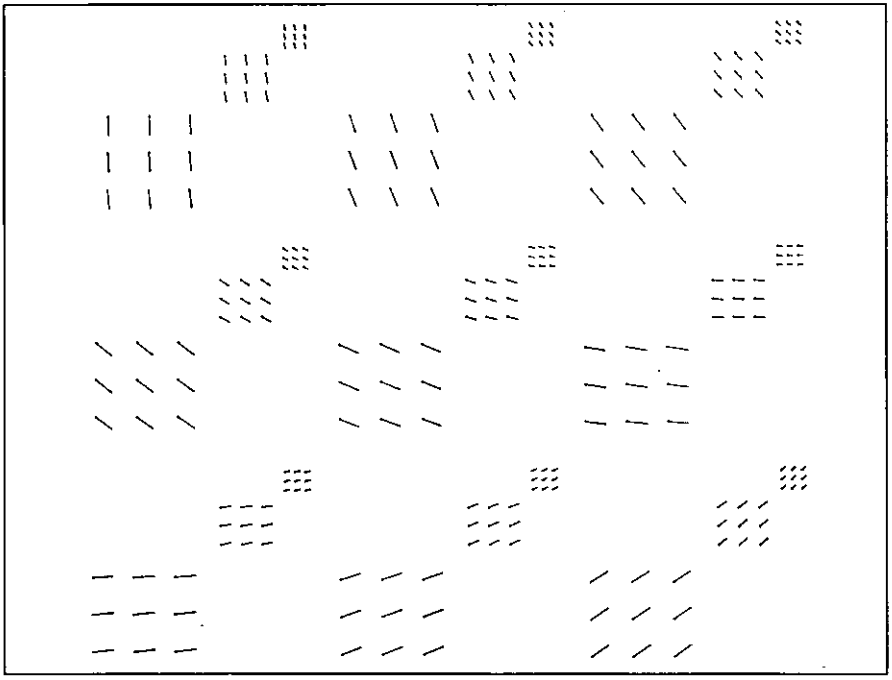
```
F-'+Oo(+w)+C'F''

'FACT'+AT=BTxT(ABX1V, 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.

Fig.5 — Set of Typefaces

Light Block (Roman Simplex)
Heavy Block (Roman Duplex)
Light Serif (Roman Complex)
Heavy Serif (Roman Triplex)
Light Italic (Italic Complex)
Heavy Italic (Italic Triplex)
Light Script (Script Simplex)
Heavy Script (Script Complex)
English Gothic
German Gothic
Italian Gothic

Fig.6 — Rank Five Complex Array



APL IN THE INFORMATION CENTRE

Compiled by Adrian Smith

The second Association meeting in 1984 was held on Friday 16th March and consisted of two papers and a panel discussion, all centred on the theme of the "Role of APL in the Information Centre".

Romilly Cocking of Cocking & Drury gave a talk on "Setting up a Company Information Centre" and the next article represents notes on his seminar produced by Wendy Hoare of Rowntree Mackintosh, whom I would like to thank for her pains. If Romilly Cocking disagrees with any of this transcript, there is a simple solution: send us your own text instead!

The next talk was by Fred Perkins of I.P. Sharp Associates, entitled "The Global Information Centre." The notes are mine; the same remedy applies in the event of a complaint about misrepresentation. I have not devoted much space to this most interesting talk which was nevertheless a thinly disguised sales pitch. No doubt I.P. Sharp Associates will oblige with glossier and more detailed documentation on demand.

Finally a most interesting panel discussion ensued which I have attempted to summarise in the dialogue that ensues.

SETTING UP A COMPANY INFORMATION CENTRE

by Romilly Cocking

Definitions and Objectives of an Information Centre (IC)

A straw poll of the audience revealed that 10 out of 80 felt they knew what an Information Centre was and 12 out of 80 felt their view was based on the IBM concept!

Romilly Cocking began with a couple of rather cynical definitions of an IC, namely:-

"The best sales aid IBM has developed to date"

and

"The DP department pacifier for irate users" — to distract their attention from applications that are late, wrong or way above budget.

A more positive definition put forward the purpose of an IC as providing the hardware, software and support to enable all employees to satisfy their information needs by being able to:-

a) get at information

and

b) use it

without becoming DP professionals.

Romilly Cocking (RC) stressed the need to keep in view the benefits to the organisation of providing information, which may be one or more of the following:-

a) To help directly or indirectly to increase sales

b) To help directly or indirectly to reduce costs

c) To improve cash flow

d) To meet legal or corporate requirements

The IC's contribution to these objectives would be in the form of:-

a) better decisions

b) better control

c) better planning

and in helping communicate these decisions to people and justifying them.

The IC may also contribute to greater efficiency by enabling fewer people to do the same work.

Justification for an IC

The biggest problem, when deciding whether to set up an IC is that of cost justifying it in numeric terms. It is impossible in many cases to demonstrate savings. RC proposed that it should be viewed as going into a separate business, with the associated risks and suggested taking a good look at the experience of other companies before doing so.

Once established, it should be operated as a business with a separate cost centre from the rest of the DP department, paying its way within the organisation as a bureau service.

IC Personnel

RC stressed that the IC would be a service business, with people as its key resource, not the hardware. The level of staffing is an area of contention and the audience were invited to make suggestions as to an effective ratio of support staff to users. Opinions were varied, ranging from 1:4 up to 1:70. IBM apparently recommends a ratio of between 1:30 and 1:100 which in RC's opinion, would be very unsatisfactory if trying to balance desirability against practicality. A more realistic and effective ratio would be somewhere between 1:5 and 1:20. Anywhere below that would be merely providing access to hardware, rather than providing an Information Centre.

Of the personal skills required within an IC, he suggested that some people would need to be salesmen, others highly skilled technically and all should be good at handling people. Furthermore, experience in the form of a "veteran" from a bureau is almost a necessity.

Software

Assuming that the quality and level of staffing is right, the next most important factor is software. Great care must be taken in choosing software that is easy to use and allows users to get at corporate data and process it. It is difficult to change products once users have become familiar with them. They will be reluctant to re-train and therefore a long term view should be taken when selecting software.

On the role of APL in the IC, Romilly Cocking's view on VS APL (as opposed to other APL's) was that it was adequate but not ideal due mainly to its restricted file handling capabilities. APL on its own would not meet all of the information requirements and has proven not to be an end user language. The majority of end users don't want to learn it and favour the "point and grunt" approach which ADRS and ADI have attempted to provide. In RC's opinion, he doubts that APL is ideal for spreadsheet type applications.

He had little to say on other software products, but mentioned the "IBM approved" packages Wizard and Intellect as looking good and being purchased by several companies.

Hardware

RC emphasised the benefits of having a completely separate mainframe from that used to process the operational systems. He warned that CPU usage would grow rapidly and that there would inevitably be a battle over charges if a machine were shared. This would result in a clamp down on usage by the IC and the consequent need to turn away customers. Costing would be much easier on an independent machine.

On the question of size, he warned that anything smaller than, say, an IBM 4341/G2 would soon be inadequate.

VM would be preferable to TSO and CMS should support a larger user base.

As well as a mainframe, the IC should support a network of VDU's and PC's with provision for hooking up PC's with the mainframe. Transferral of data between PC and M/F as yet presents a problem, with various partial solutions but no satisfactory universal one.

Operating the Information Centre

Education, consultancy and support are among the responsibilities of an IC and Romilly Cocking suggested issuing a newsletter, arranging seminars and setting up a help desk as just some ways of meeting them and making the IC more effective. IC staff should spend time with users, find out how they operate, since it may not be obvious to them how the IC can help. The IC may become involved in developing software with users, which RC says is a good thing. If users are encouraged to develop systems entirely independently, they will usually either fail completely of if they are successful, will be tempted to tackle larger systems and will inevitably meet and try to solve the same problems that DP met and overcame years ago. IBM maintain that the IC should not become involved with development, but one of the original reasons for setting up an IC was to help clear the backlog of systems waiting to be developed and implemented. The IC can and should take on some of the small developments and complete them more rapidly.

Summary

RC summed up by stating that the IC concept has worked and that is possible to identify specific projects which have been cost effective. It gives management a tighter grip on individual developments and ensures that direct savings can be achieved.

If the IC is set up properly, there will be rapid growth and it is essential to sort out the charging system right from the start. It is much easier for the IC to establish whether and how it is contributing if each project is costed out. Therefore, when sales revenue grows rapidly, there will be ample justification for expanding its resources.

The talk ended with the questions "Is the IC a revolution in computing and should it be staffed from outside DP?" RC maintained that there are often revolutions in mainstream DP and that the Information centre needs the continuing skills from it.

THE GLOBAL INFORMATION CENTRE

by Fred Perkins

This talk also began with a rather cynical view of information centres and leading up to it was a short history of their beginnings. In the mid-to-late 70's, at a time when users and DP departments were growing apart, backlogs were getting bigger and users were getting angrier. IBM came up with the concept of the Information Centre and marketed it extremely well. They gathered together a collection of applications software, which, in the hands of the users would put the emphasis back on them, with support being provided by DP. The outcome? — "Never have so many waited so expensively for so little" — according to Fred Perkins. He cited some of the problems associated with IC's:-

a) user isolation

b) lack of tools to help make systems portable

c) Growth/migration problems

d) tools not part of the IC concept

e) incompatibilities between mainframe and PC's

f) packages in different languages

g) inability to access data outside the company and across different systems.

The rest of the talk was devoted to the IP Sharp Global Information Centre. It claims to overcome all of these problems mainly due to the following:-

a) Single development language — Sharp APL, identical on the PC to that on the mainframe.

b) Worldwide communications network, with access to the world's largest on-line library of numeric data bases.

c) Sophisticated applications software products.

d) Full support for a wide variety of hardware devices.

PANEL DISCUSSION ON INFORMATION CENTRES

RC	Romilly Cocking	(Cocking & Drury)
MG	Mark Griffiths	(APL consultant)
TM	Tony Moore	(ex Tesco)
FP	Fred Perkins	(I.P. Sharp Associates)
DP	David Preedy	(Imperial Group)
RW	Richard Watkins	(CACI)

Q Indicates a question from the floor
?? Indicates an unattributed comment from the panel or the floor

Q: How do the so called '4th Generation Languages' fit into the Information Centre strategy?

RC: Can you be sure they will work 5 years on? IBM have landed plenty of people in the cart with VSPC — they could just as well do it with FOCUS if they chose.

FP: "APL is the only language anything like a 4GL!" (James Martin); what does 4GL mean anyway, and are such languages capable of wide adoption?

RW: They are Report Writers to deliver tools to a user in a way which does not distract him from the task in hand. They are tending to move to more 'English-like' syntax, and products like Focus certainly have a long life ahead of them.

MG: 4GLs are still a DP-oriented product. APL allows users to do their own thing; 4GLs are productivity aids for DP (RW: yet the salesmen aid them at users?!)

TM: If NOMAD, SASS and the rest don't run in 5 years' time, a lot of users really are in trouble!. Many companies already have a variety, and find it hard to link them — APL can come in handy as 'glue'.

DP: This proliferation may be a barrier in itself — maybe a genuine English 'front-end' will soon be necessary?

Q: There is still a DP backlog, with many projects not being tackled .... why??

RC: Faster hardware hasn't helped much; the human resource is still the limiting factor. We should look seriously at APL even for very large DP systems (Gigabytes upwards).

FP: Programming may be quick in APL, but for big projects the programmer's time is only a small! fraction; often APL programmers are lousy at project management, which doesn't help. If the 4GLs don't deliver the trend will continue — the user demands function, not a set of arbitrary rules to learn. No-one bothers how VisiCalc works — they just see it!

Q: What about data; do we provide duplicate copies in all sorts of different formats?

FP: The user doesn't care. He needn't know where the data is, and in general you should convert it when needed; you can't keep 6 copies of a real-time database up to date.

TM: At the moment there is no satisfactory answer on IBM systems. Maybe DB2 will help ....?

RC: ... it probably will, but golly will it mop up the CPU!

RW: If you have spent 10 years building a set of systems the physical structures are likely to be totally wrong for the IC. Copying may be the only short-term solution.

RC: VS APL finds data access and resource control hard — hence the recommendation to put it on a separate machine. Sharp APL can share files sensibly on the main machine.

FP: 4GLs have this problem too — problems of double updating and integrity can still force seperation and copying.

Q: What effect will the Macintosh have? It looks more suitable for a coffee table?!

TM: People want windowing, but no-one quite knows how it will work with large volumes of data.

??: Nothing an IBM 3279 can do remotely resembles the Macintosh.

??: Are touch screens a gimmick — after a month people prefer to type anyway because it's a lot quicker?

??: The Sunday supplements have a lot answer for — they have boosted expectations for hardware and increased demand, but done nothing to increase resources.

??: Some of us are just as guilty. In the hands of a real expert APL is a superbly powerful modelling tool — much faster than the 4GLs. Hence a sniff of it raises the user's expectations, duly increasing that DP backlog!

COMMUNICATING WITH APL

Compiled by Adrian Smith

This meeting took place in London on 27 March 1984, and featured three speakers.

Richard Nabavi of MicroAPL described an example of a mixed Ethernet network of local microcomputers. He has thoughtfully documented his seminar and we reprint this overleaf.

John Pym of Inner Product showed how Prestel could be used as a publicly available networking system. His theme was suspiciously similar to his paper for last year's APL Business Technology Conference, but we enjoyed his brand of offbeat humour so much that we've included his paper, after one or two tidying up operations.

Finally Tim Perry of APL People spoke about the use of APL within IBM, a talk that I have documented based on my notes of his presentation.

NETWORKING APL MICROCOMPUTERS

by Richard Nabavi

What is Networking?

Of all the buzzwords in the computing industry, 'Networking' is the one which is the most misused, and yet the most indicative of underlying user needs for computing in the late '80s. Just as in the early seventies, 'Interactive' and 'Real Time' were regarded as the adjectives to look for in assessing computer systems, today manufacturers all claim that their systems have a networking capability.

In reality, networking can mean different things. Looking through the computer magazines, there seem to be three main ways in which the word is used (or misused). These can be described as Point-to Point, File Serving, and True Networking.

Point-to Point Networks

Many vendors of 'networking' products are in fact talking about something very simple. This is illustrated in Figure 1, and a typical example of such a system is Clearway, a product of Real Time Systems Ltd. Essentially, what this consists of is a coaxial cable (or, more generally, any means of transmitting messages, such as a fibre optic link) into which a number of devices can be linked. Typically, the devices which will be hooked on to the cable will be terminals, printers and computers. The idea is that, instead of wiring terminals and other peripherals to computers via ordinary 'RS232' cables, the network is used instead to provide a multitude of point-to-point links between the various devices. In other words, this sort of 'network' is really doing nothing except making neater the connections between devices, which otherwise would be connected via a mess of independent cables.

The advantages of a system such as Clearway are obvious. As new devices are added, you do not need to rewire your building. A single terminal can switch between different computers under software control, instead of having to replug RS232 cables every time you wish to log on to a different computer. This is a lot more efficient than conventional wiring, plugboard and switches, but these sorts of systems should not be confused with networking of computers, since they do not allow the sharing of data between different CPU's connected to the network, They represent a very useful simplification of the problems of lots of data cables around a building, but they do not involve any radically new approaches to data processing or the way in which computers are used.

File Serving Networks

Many of the claims which computer vendors make for networking capability refer to something which again should not be confused with true networking. This is what I call the File Server type of system. The philosophy behind this is again fairly simple, and can be very useful. The 'network' is built around a single central computer, usually with a hard disk and perhaps a printer and other peripherals connected to it, into which are connected (in a 'star' configuration) external satellite computers (see Figure 2). These satellite computers will usually be small microcomputers, with no disc facility at all (or perhaps with a small amount of floppy disc storage). The idea is

that hard disc storage is relatively expensive, and that users in many applications will wish to share common data. Note that, in this type of system, the computers in the network fall into two distinct classes — File Servers (of which there will usually be only one), and Satellite Units.

A number of points should be apparent. The first is that only the File Servers need to have an advanced, multi-tasking operating system. This is because the Satellite Units are supporting only one terminal, and, when a request is made, for example, to read a record from a file, the Satellite Unit which made the request will wait until the record is returned as data by the File Server. The second point is that the expansion capability of the network is limited by the response time of the central File Server. Essentially, we have only one computer serving all the disc read/writes for everyone on the network, which is not much better than having only one multi-user computer in the first place. Thirdly, although there may be peripherals such as printers or graph plotters attached to some of the Satellite Units, these will not be accessible to other Satellite Units. In other words, we are re-creating the concept of a central computer — comprising disc files and peripherals — which is accessed by remote users, who may or may not have some local printing facilities. For many of us, it was to get away from the restrictions of centralized computing that we first turned towards microcomputers, and the re-creation of a central computer is like bringing back the bath water with the baby.

True Networking

By now it should be becoming clear what I mean by a true networking computer system. The characteristics are:

- The ability to connect together a number of computers, each logically equivalent as far as the network is concerned;
- Each of the computers on the network to be single or multi-user as is most convenient;
- Each of the computers to have the possibility of disc storage, although the amount and type of storage will vary between the different machines;
- Each of the computers in the network to have the possibility of being connected to a number of peripherals;
- Any user on any machine on the network to be able to access (transparently to the applications software) any file on any disc on any machine in the network (subject of course to normal security checks);.
- Any user on any machine on the network to be able to access any peripheral on any machine on the network;
- Full multi-user controls including file and record locking to apply across the whole network.

Further features which in practice we have found desirable are:

- The ability for any user to initiate a task on a remote machine, either attached to his terminal or to run as a background task. A good example where this is useful is to start a print spool job on a remote machine;
- The ability to connect simultaneously to several networks;

Figure 3 shows a typical small configuration, which is in fact MicroAPL's in-house network system.

Operating System Support of Networking

Most discussions of networking concentrate on the characteristics of the physical connection between devices on the network. Unfortunately, most computers in use today are based on operating systems which were conceived before networking was developed. This means that networking has to be added on as an 'extra', and is not built-in to the design of the systems software. The consequence of this is that limitations have to be imposed on what you can do with the network. For example, some major manufacturers have announced Ethernet connections between their computers, but when you come to examine the capabilities in detail, you find that the only way in which data can be accessed on a remote machine is by copying the whole

Fig.1 — Point to Point Networking

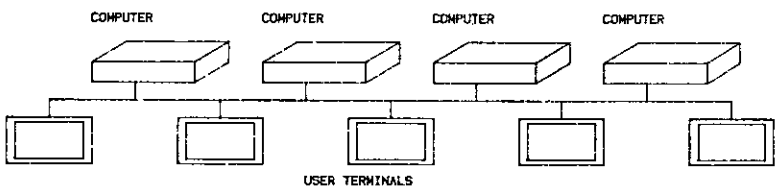


Fig.2 — File Server Networking

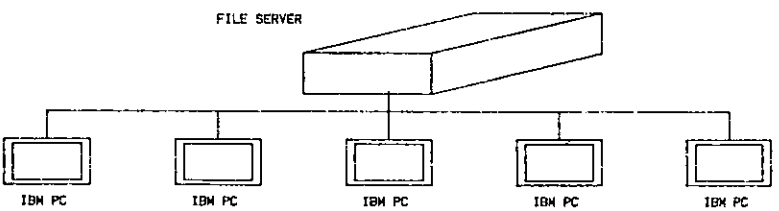
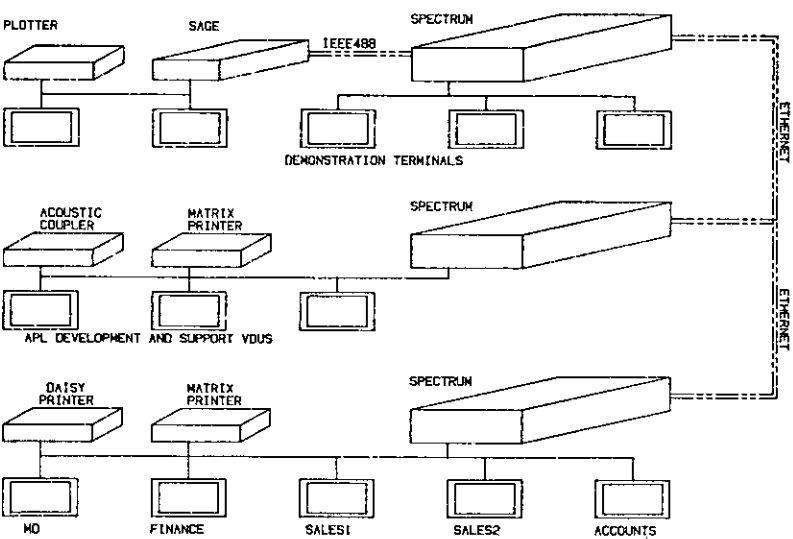


Fig.3 — MICROAPL In-House Network



file from the remote machine to your home machine.

In order to illustrate the way in which networking can be built in to the system software of a modern computer, let us look at MIRAGE, one of the few operating systems which was designed right from the beginning to cater for network operation. MIRAGE is a multi-user, multi-tasking operating system which was written by a UK company, Swifte Computer Systems Ltd, for Motorola 68000 computers. It has been in use since 1981 on MicroAPL's range of 68000-based micros, and it implements networking in a very simple way.

To understand the way in which MIRAGE handle network file accesses, let us first consider how the operating system refers to a file on a single machine. Files are organized in 'directories' on logical discs on the system. For example, suppose a user wanted to type out the contents of a text file called 'SYSTEM.HELP' which was sitting in the 'DOC' directory on logical disc 3 of his machine. Using the MIRAGE utility 'TYPE', he would enter at the keyboard:

```
TYPE DSC3:SYSTEM.HELP[DOC]
```

'DSC3:' refers to the name of the logical disc on which the file resides, the directory in which it resides is entered between square brackets, and the file name is divided into a main file name ('SYSTEM') and a file extension ('HELP'). As a convenience, it is not always necessary to enter all of this information, since each user has a default disc and directory into which he is 'logged', and which he can change at any time. For example, by entering:

```
LOG DSC3:DOC
```

he can change his default directory to be DOC on DSC3:. If he then enters:

```
TYPE SYSTEM.HELP
```

the operating system will by default look for the file in the correct place.

The extension of this to a networked system is simple. As well as specifying the directory and disc on which a file is to be found, you specify a machine on the network. This is done by adding to the front of the file name a 'Node' name ('Node' means machine on the network). This is distinguished from the logical disc name by being terminated by two colons rather than one. The home machine is always 'SYSO::'. Thus, in the example above, the user could if he wished have used the full file name:

```
TYPE SYSO::DSC3:SYSTEM.HELP[DOC]
```

This means: Type out the file SYSTEM.HELP, which is to be found in directory DOC on disc 3 on my home machine. To access a file on a remote machine, you simply change the Node name to that of another machine on the network. For example, a user on a remote machine who wanted to type out the same file might enter:

```
TYPE EXT4::DSC3:SYSTEM.HELP[DOC]
```

where EXT4:: is the name (as viewed from his computer) of the machine on which the file resides. Naturally, it might become rather tedious to have all the above information for every file reference, so you can log into a remote directory. For example:

```
LOG EXT4::DSC3:DOC
```

would cause the default node, disc and directory to be the ones we were interested in.

Another couple of examples should give an idea of the possibilities. To copy a whole file from a remote machine to your own:

```
COPY SYS0::DSC0:MYHELP.DOC[HELP] = EXT4::DSC3:SYSTEM.HELP[DOC]
```

This copies the file SYSTEM.HELP from the DOC directory of disc 3 on remote machine EXT4:: to the HELP directory of disc 0 of the user's home machine, renaming the file to MYHELP.DOC in the process.

To copy a file from one remote machine to another remote machine:

```
COPY EXT6::DSC0:MYHELP.DOC[HELP] = EXT4::DSC3:SYSTEM.HELP[DOC]
```

This is the same as the previous example, except that the file is being copied from remote machine EXT4:: to remote machine EXT6::.

High Level Language Support of Networking

Clearly, all the above facilities are of limited use if they cannot also be used by high-level languages such as BASIC, Pascal, APL, and so on. Equally, in a true networking environment, applications software such as word-processing software and spreadsheets should be able to access files anywhere on the network. This again shows how important it is for networking capabilities to lie right at the heart of the system software, so that high-level support for these facilities should 'fall-out' automatically without special arrangements. We can see how this works by considering how APL.68000 under MIRAGE can access files across the network.

Let us first consider how APL finds files on a single machine. The user might, for example, enter:

```
)LIB
```

which would print out his default library of APL workspaces. He can also access up to 9 other libraries, by (for example) asking for:

```
)LIB 5
```

What in fact happens here is that the APL interpreter maintains, for each user, a table (technically an APL system function Quad-MOUNT) of his current ten libraries. By default, these are logical discs 0 to 9 on his home machine, with his default MIRAGE directory. Thus, the default table looks like this:

```
SYS0::DSC0:JIM
SYS0::DSC1:JIM
SYS0::DSC2:JIM
.....
.....etc.
```

At any time, the user (or the applications program) can modify any or all of the rows of this table. For example, the first three rows of the table might be changed to:

```
SYS0::DSC0:JIM
SYS0::DSC4:MARY
SYS0::DSC5:LUKE
```

This would mean that library 0 would be directory JIM on disc 0, library 1 would be directory MARY on disc 4, and library 2 would be directory LUKE on disc 5.

The extension to a networked configuration is obvious and very simple. You simply specify the node for each logical library. For example:

```
EXT3::DSC0:APL
EXT6::DSC3:SUE
EXT9::DSC2:DAVE
```

In this example, libraries 0, 1 and 2 are all on different remote machines.

For those familiar with APL.68000, note that as well as workspace loading, saving, and copying, the Quad-MOUNT table also specifies the location for component file accesses using APL.68000's file system. Up to ten different nodes/discs/directories may be specified using Quad-MOUNT. If this is not enough, you can change Quad-MOUNT dynamically under program control to give access to an unlimited number of file locations. In addition, print spool files may be directed to a remote machine, as may arbitrary input/output, so that, for example, a graphics package may produce graphs on a flat-bed plotter which happens to be connected to a different computer.

The important thing to appreciate here is that an application program, written in APL, need not be modified to run off a file on a remote machine. Suppose that, for example, we have written a management information system which was originally designed to run on a single machine, with files on DSC0: and DSC1: of that machine. Later on, we link the machine into a network, and we want users on any machine to be able to access the same files, without changing our applications software. All that is necessary is for the above table (the 'Quad-MOUNT' table) to be initialized to set up the first two rows for access to the machine on which the files exist, and then the user will be able to run the application without any changes. The initialization can of course be automated so that the user is unaware that it is happening, if desired.

Access to the network via other languages is similar; essentially, we merely change the node part of the file name to go to a remote machine. To run the word-processing software off a remote disc, you simply LOG into the remote directory before running it. With the single exception of one piece of software which maintains memory-resident disc buffers, all programs written under MIRAGE can run unchanged across a network.

File and Record Locking on a network

Probably the most embarassing question you can ask a supplier who claims to support networking is whether file and record locking operate across the network. The importance of this question will be obvious to anyone who has experience in writing multi-user applications software, since by definition networked systems are multi-user. A quick example should show what we mean.

Suppose we have a stock control system, and the files can be updated both by the Goods Inwards department and by the Goods Outwards department — which seems reasonable. Consider the case where a consignment of 100 Size 10 Wonderflanges arrives, and at about the same time an order for 18 of the same item is being despatched. The stock level before all this happens is 134.

The Goods Inwards people go to their computer terminal and call up the stock record for the Size 10 Wonderflanges, and at the same time the Goods Outwards people also do the same. Goods Inwards type faster, and so they enter the arrival of 100 items, to make a stock level of 234, and the record is written back to the file. Meanwhile, Goods Outwards see that there were 134 items in stock, and they enter their Goods Out details to reduce the stock to 116, and this record is written back to the stock file, overwriting the record which Goods Inwards have just written. The final result is that the stock level will be erroneously recorded as 116 instead of 216.

This is an absolutely standard problem in multi-user systems, and careful consideration will show that there is no way to avoid the difficulty unless the operating system maintains some form

of file or record locking feature. What happens then is that, before updating the file, the record is 'locked' and no other user can access it until the update sequence is complete. Depending on the sophistication of the operating system, this lock might be on the whole file, or on specified records, and it might be a lock against all accesses, or only against writing. Whatever arrangement is used, a commercial multi-user system must have some form of file locking if multiple on-line file updating is to be possible.

Exactly the same considerations apply to networked systems. If true networked processing is to be possible, any user must be able to lock any file on any disc on any machine on the network (again, subject to security considerations). MicroAPL's systems can in fact provide file and record locking across the network, as well as providing two levels of lock (read only or read/write). Again, the fact that the lock is occurring over the network is transparent to the applications software.

Case Study No. 1: Imperial Group

So far we have considered the facilities offered by MicroAPL's networking software in theory. How does it work in practice?

One example of a network used by a MicroAPL client is a group of large MicroAPL Spectrum supermicros installed at the London headquarters of Imperial Group. These systems serve over thirty users, with a large number of different printers, graph plotters and other peripherals attached to the machines on the network. The multi-user machines are used for two main (related) applications:

— Consolidation of the management accounts of the various companies in Imperial Group to provide an overall set of Group accounts.;
— On-line display of thousands of graphs detailing performance of the different parts of the Group against budget, previous year's performance, etc. This is done on Tektronix colour screens installed in the boardroom and in the offices of the senior executives of the Group. Hard copy of graphs is produced on Hewlett Packard flatbed plotters.

This installation is an example of the use of networking to create a large effective logical computer out of a number of smaller machines. Initially, two separate machines were purchased (one for each of the above applications) and data transfer between the machines was via magnetic tape cartridge. The initial two machines were gradually expanded with extra memory and extra I/O ports, and then linked together using an Ethernet link to facilitate exchange of data between machines. Subsequently, further processing power was added by linking in another machine with a larger disc capacity. At the time of writing, the installation has been built up to an overall system of 48 RS-232 I/O ports, approximately 8 Megabytes of RAM, and 144 Megabytes of disc capacity, with two tape drives for back-up and data exchange.

A number of benefits accrue from this approach. Extra capacity in terms of CPU power, RAM, disc capacity, I/O ports and peripherals can be added as and when it is needed, with no need to change software as the installation grows. The use of several independent computers on a network gives the advantage of some degree of fault tolerance whilst permitting the sharing of common data and peripherals. Data can be backed up on to a remote machine, reducing the need for expensive tape units and simplifying backup and recovery procedures. New releases of software can be tested on one of the machines on the network, with access to the real data, whilst maintaining the previous version on other machines on the network. System maintenance — both in hardware and in software terms — is easier than it would be on a single minicomputer of comparable overall performance.

Case Study No. 2 — BASF

The Imperial Group system is an example of a tightly-coupled set of machines; the only practical alternative to their network of supermicros would be a minicomputer or small mainframe. A different

type of installation — built out of similar building blocks — is to be found at the large BASF complex at Ludwigshafen in Germany.

BASF have purchased a number of MicroAPL Spectrum machines of different configurations. These include machines with 5" and 8" floppy discs, 5" Winchesters, 36 Mb 8" Winchesters and 72 Mb 8" Winchesters. The systems are used for a variety of applications and are scattered about the very large site. Some are used in multi-user mode, others are single-user dedicated systems. Devices connected to the systems vary between ordinary terminals and printers to laboratory instruments. The systems are used for scientific and engineering work, as well as straight commercial work.

In this case, an Ethernet cable of 1.5 kilometres has been used to link together the various machines. The purpose here is to allow simple exchange of information (even though some of the systems are of different configurations so that they cannot, for example, swap floppy discs to exchange data), and to avoid the isolation of having separate computers in various locations. Whereas in Imperial Group the effect of installing networking has ben to create effectively one large logical computer, here the different computers are exchanging data less frequently and are not necessarily running related software. The installation of the network has meant improved communications, easier exchange of programs in a development environment, and access across the network to any peripheral, disc or tape on any machine.

The BASF network is thus an example of a network of powerful micros which could run independently, but which form a much more flexible set of systems when linked together. The alternative would have been to stick with a collection of non-networked microcomputers, with all the disadvantages of isolation which that solution can cause. The problems of non-linked machines would have been particularly bad in view of the size of the Ludwigshafen site; whereas in many cases it may be acceptable to walk from one machine to another with a floppy disc if you want to exchange data, when the other machine is a mile away you do not want to do this too often.

The real challenge — true multi-vendor networking

The systems we have been talking about so far are very flexible, but are limited in one very crucial respect. This is that the computers connected to the network have to be running the same operating system. Thus, you cannot simply take an assortment of different computers from a variety of manufacturers and link them together.

True multi-vendor networking does not exist at this moment. Whilst a number of specialist suppliers have worked on bridging this gap, the solutions offered for multi-vendor networking have to be very restricted. This is not surprising, since the facilities offered by different operating systems — let alone how those facilities are implemented — vary enormously. Therefore the ideal of being able to access files on any machine on the netwok, irrespective of the make and model of the machine, is very far off.

Worse than this, even if you could load a program from a remote machine into a computer of a different type, this would still only be of limited usefulness. Try to run a program written for the 8088 chip in an IBM PC on the 68000 processor in an Apple Lisa and you will not get very far. Even pure data files will be of different formats for different machines, with the possible exception of text files which are fairly standard. Even Unix does not provide machine independent program and data files, because of the multiplicity of different Unix versions and the fact that many programs are compiled into the native code of the machine they run on. The establishment of accepted vendor-independent standards for connection of systems is still at a fairly early stage, although some progress has been made.

Having said this, at MicroAPL we have looked into the question of providing at least some

machine-independent networking facilities, by effectively emulating the MIRAGE operating system network interface on different machines. This would at least provide access to data on various remote machines, although the applications program might have some conversion to make use of the data. This work is only at a very preliminary stage, but we have at least established the technical feasibility of the approach, and we hope to work with a couple of our existing customers over the next eighteen months to set up a working system providing access to various mini- and micro-computers.

Conclusion

Networking of small computers is still at an early stage of development, but customers today can already reap very real benefits from the work that has already been done. The main obstacles to development are the lack of general standards for data exchange at the file level, and the fact that most operating systems in use on small computers were not conceived with networking in mind. The term 'networking' is used in many different senses, and vendors' claims should be examined carefully to establish exactly what facilities are offered, and what software support they require.

INTERFACING VIEWDATA AND APL

by John Pym

"Learning is learning not to think about operations that once needed to be thought about. We learn to make processes of deliberate thought instinctive and we learn to make automatic and instinctive processes the subject of discriminating thought." *Sir Peter Medawar.*

Viewdata Systems

Viewdata systems provide a simple and effective means of accessing a large database of information. One or more mainframe computers contain thousands of "pages" of text. A page of information represents a string of text that will fit on a standard television screen, 24 rows by 40 normal characters or graphic symbols. The mainframe computers are linked to the user of the viewdata system by telephone via a modem to a display device. This display device may be either a standard television, VDU or microcomputer screen. The user accesses the viewdata system by dialling a local number, or if the modem is "smart"by pressing a key on the modem to dial the number, or if using a microcomputer by having the software program the modem to dial the number. The user signs on to the viewdata system by typing an identification and a password. This again may be done automatically by the software on the microcomputer, in which case the password could be checked by the software on the microcomputer. Once signed on the user may access any page directly by typing the page number or from any given page specify a number from 0 to 9 which determines the next page to be shown. Not all pages need provide the full ten choices. The definition of which pages will be connected to the current page is made by the information provider.

In fact the pages held by the mainframe computer form a network in which it would be possible to access any one of a million pages directly with only eight keystrokes, or be directed through a sequence of just six directory pages, just six keystrokes, to any particular page. It is possible to be directed "down" to pages providing more detailed information on a particular topic, but you can then be directed back "up' by a different route from the one you came "down".

The standard viewdata modem sends to the mainframe computer at 75 baud but receives from the mainframe computer at 1200 baud. This means that sending is fast enough to cope with the speed of the user typing the few characters needed to request the next page, but the response time required to send the thousand or so characters representing the page is small. In addition the system operates in full duplex, so that the user may request the next page even if the system is in the middle of displaying the previous page. so that to see choice 2 from the current page, choice 8 from that page and choice 3 from that page, entering 283 will immediately display the last page and omit display of the two intermediate pages.

Microcomputer/Viewdata Interface

The microcomputer can be programmed to act as a television with viewdata attachment, in which case it will allow the display of pages in exactly the same fashion as the television system does. But given the ability to program the microcomputer it is possible to provide additional facilities:

a) Save pages on disk.
b) Recall pages saved on disk when off line.
c) Display two pages at once.
d) Changing resolution of display.
e) Programmable baud rates.
f) Programmable user identification.
g) Interface to other packages on the micro.
h) Interface to other languages on the micro.

The last facility is of interest as it is posible to make the information contained on a page available for further processing. Currently two interfaces are available from Inner Product, one to BASIC and one to APL.

APL/Viewdata Interface

The interface to APL consists of a single APL function. The left argument is a numeric or character string and the right argument is a function number. The functions available are:

a) Set baud rates for send and receive.
b) Send character string to viewdata system.
c) Send character string to viewdata system and await reply.

The result is returned as a character string which may then be saved or manipulated in any way desired using APL.

Interfaces

The ease with which it is possible to scan the vast amount of information on a viewdata system prompted me to consider the interface between the user and the microcomputer presented by APL, systems written in APL and other microcomputer packages. This is the relevance of the quote at the start of the article, as after many years of using solely APL to implement systems, I have over the last few years been using other computer languages, systems not written in APL both on mainframe computer systems and on microcomputer systems. The consequence has been a reexamination of some assumptions and habits acquired over the years. BASIC can seriously damage your health but so can any prejudice.

Some prejudices in favour of APL were merely confirmed. For instance, many other languages either do not have a consistent syntax or are internally inconsistent or both. Inconsistent syntax means that it is not possible to deduce how statements should be phrased from what is already known about the language. For example, rather than a function producing a result which can either be displayed or can be assigned to a name or a file, some functions will display, some can only be assigned to a name and some can only write to file. It is of course possible to design APL systems with inconsistent syntax but there is less excuse. Another example that appears in many languages is the treatment of empty character strings. Many do not allow them; a character string must have at least one character in it, which can be extremely irritating if you wish to distinguish between a null response and a blank.

In other areas APL systems come off less well by comparison. One of these is in full screen interfaces. Historically, APL systems have been designed round fifteen character per second Selectric typewriters and many systems seem to be designed as if this was still the form of access. Even those that take advantage of full screen still do so in a static mode, whereas a quick look at any standard spreadsheet or word processing package confirms that a much more dynamic and responsive interface is possible.

Languages

The first thing most people do with APL is to write a "language" of functions appropriate to the application under development. A chisel is fine, but a table and chairs are more useful when giving a dinner party. A Swiss army knife is fun to have, especially if you find a horse with a stone in its hoof, but it's easier to eat dinner out of bowls. APL is fine for developing such languages but already enforces constraints on the type of syntax that is possible if APL functions are used, rather than writing a syntax analyser in APL. For example: right-to-left order and a maximum of two arguments per function. A look at other languages will demonstrate that there are other possibilities. For example, spreadsheets provide a language that essentially gives a

function with as many arguments as you like, up to the number of cells in the spreadsheet. Such a language represents a "non-algorithmic" method, not a sequence of steps but a sequence of states. It is quite possible to give spreadsheet solutions to problems that at first sight would seem unlikely candidates for such non-algorithmic methods. Good examples include the Tower of Hanoi problem and the eight-queens problem; see Scientific American "Computer Recreations" section, for October 1983 and January 1984. Other languages such as Smalltalk provide examples where the function depends on the "class" of its arguments, thus generalising the idea sometimes used in APL of the same function doing different things depending on the type or rank of its arguments.

Functional design

It is extremely difficult to forget how a problem might be solved using a particular programming language or a particular package chosen from the ones that you happen to be familiar with. Trying to sit back and design how a system will look and feel without having some part of you saying "how the hell can you achieve that effect with a Dodo Megabit Mark IV computer and the 'Obvious' data base language" is hard. But the means for achieving Visicalc were available a long time before the program was actually produced.

Conclusions

As microcomputers become more common, APLers should not remain parochially confined to the way things have always been done when communicating with APL, but should look outside APL to other languages and other systems for new ideas and methods. Crossfertilisation can produce offspring better suited to a new environment.

IBM-BASED APL COMMUNICATIONS

by Tim Perry

The specific example Tim used was of an IBM factory with around 2000 employees, and 1300 terminals. A combination of APL and VM/CMS was expected to support a growth path like this:

		1983	*1984*	*1985*
Mainframes	:	3	5	3
Versions of APL	:	3	4	1 (APL2)
Users	:	200	400	600
PCs	:	2	35	70
Cost/Unit	:	100	50	25
Units/hr	:	100	180	250

i.e. a four-fold reduction in the cost of a unit of computing, with a similar increase in the amount of useful computing achieved per logged-hour. The use of VM/CMS was seen as vital, in that it gave APL a consistent environment across the board from the PCs to the biggest mainframes, and thus facilitated communication between APL systems and also between APL users. If need be APL could use the power of CMS or CP to send messages worldwide, look up someone's address in the global phone directory, or access a host of other useful software which Tim could only hint at (it being somewhat internal to IBM).

Tim dropped a number of interesting hints about the way you can put some go-faster stripes on your APL, for example by giving it a fixed chunk of memory to run in, which all users share. This has the beneficial side effect of making all 200 CMS commands available (not just the usual subset). He also emphasised the need to restrict the user base so that good response times were always guaranteed — programs like FRANGO need to be watched very carefully lest they slow everyone down.

Other simple things you can do include:

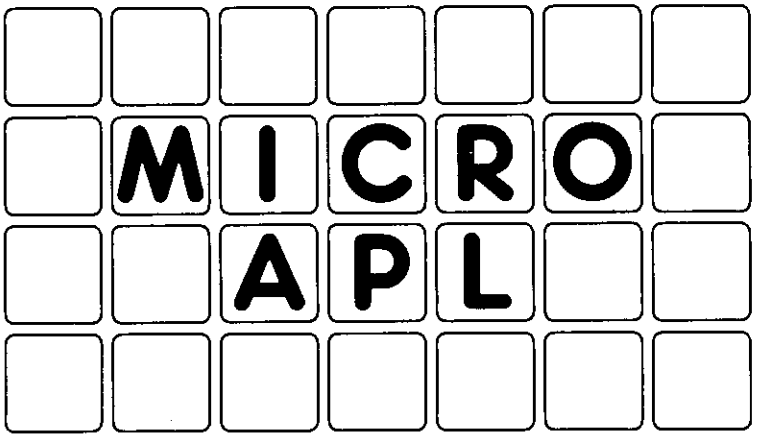
- using the CMS stack to get at the output of CP commands (such as Query Time) for genuine information on resource usage.
- making sure you use the best AP for the job in hand (generally AP110, but look for AP999 which is well hidden in the small print, and is 25 times faster than AP111).
- using PGF or (even better if you can get it) Chart for your graphics. The new AP126 lets you call Chart (or the ICU to its friends) direct, and it also allows you to call back pictures off file and get a sensible result when the FSSHOW is terminated. This lets you build a very effective (and rather cheap) Prestel imitation complete with top-quality graphics.
- going for big workspaces (1.5 to 15 Mbyte), but keeping the maximum object size below 64K.

All of which is helping to make APL increasingly useful as a communication tool for people. The use of online help and news means that any piece of useful software (be it good or awful) gets quickly spread around the system, while the availability of standard packages (ADRS/ADI) also helps users to swap data and ideas around.

In conclusion, if you are serious about the idea of management support via computers you need APL, VM/CMS (preferably plus classified bells and whistles), and someone to put quite a lot of effort into tweaking the system. Given these conditions, the resulting system can be very effective, and surprisingly efficient.

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.



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

WHY APL? A NON-TECHNICAL INTRODUCTION

Robert Bittlestone

What is APL?

APL is a general purpose computer language that many people have heard about relatively few use, at least compared to languages such as BASIC or COBOL. It has a reputation for being rather mathematical, for using funny symbols that look incomprehensible, and for attracting a lunatic fringe of dedicated converts who turn their back on using any other language. Computer experts say that it breaks all the rules in the book; it has no control structures, it has a bewildering number of ways of doing the same thing; it encourages anarchic programming; it's the "marijuana in the programmers' garden"; its a "mistake carried out to perfection".

Why then do people use it at all? The answer is very simple. In nearly all the cases I have seen where APL can be used instead of another language, it generally cuts project development time down dramatically. Jobs that typically take several weeks in other languages are developed in a few days in APL. If that kind of productivity is of interest to you — and if you would like to know how APL suppliers are reacting to the (in some cases justifiable) complaints about APL by "civilising" the language — read on.

Packages vs. Languages

If you have a job that you think would benefit from computerisation, then you have a set of decisions to take which I have tried to summarise in Figure 1. Your first action is to determine whether a suitable package exists or whether you need to develop something specific. You may know that your needs are so specialised that no package is likely to have been developed for them, or you may have surveyed the marketplace already and determined that the packages which are available are unsuitable. Let me just say in passing that APL itself is not a package (although packages can themselves be written in APL) and consequently if you can find the right package "off the shelf", you probably won't have to get involved in the choice of language anyway. I might perhaps add that if there is any possiblity of having to modify a "standard" package, the rest of the chart may have some bearing on the ease of such modification.

So you've decided to develop something specific. Will you write it within your department, will you contract the job our to the data processing department or other external team , or will you use a mix of internal and external people? The problem with contracting jobs out in their entirety is that you are the one who know broadly what you want to be done, whereas the outside people only know how the computer works. To quote a price they have to analyse your sytem. This is not straightforward because your needs are most unlikely to be well defined at this stage — at least they shouldn't be. Many observations of this principle at work led me some years ago to coin the following aphorism:

> "The requirements of a project are a result of the
> experiences you gain during its implementation."

If this is true for you, I think you're likely to find it difficult or impossible to specify your needs in advance. It may be quicker and more efficient for you to consider spending the time and money (whether real or as an internal charge-out against your departmental budget) that you were going to allocate to the outside team on training your own people on how to use the computer instead. Let the transfer of information go the other way for a change. Let yourself be the one to ask all the questions this time. That way your department will acquire very valuable skills which will be useful for all sorts of future jobs.

Fig.1 — Packages vs Languages

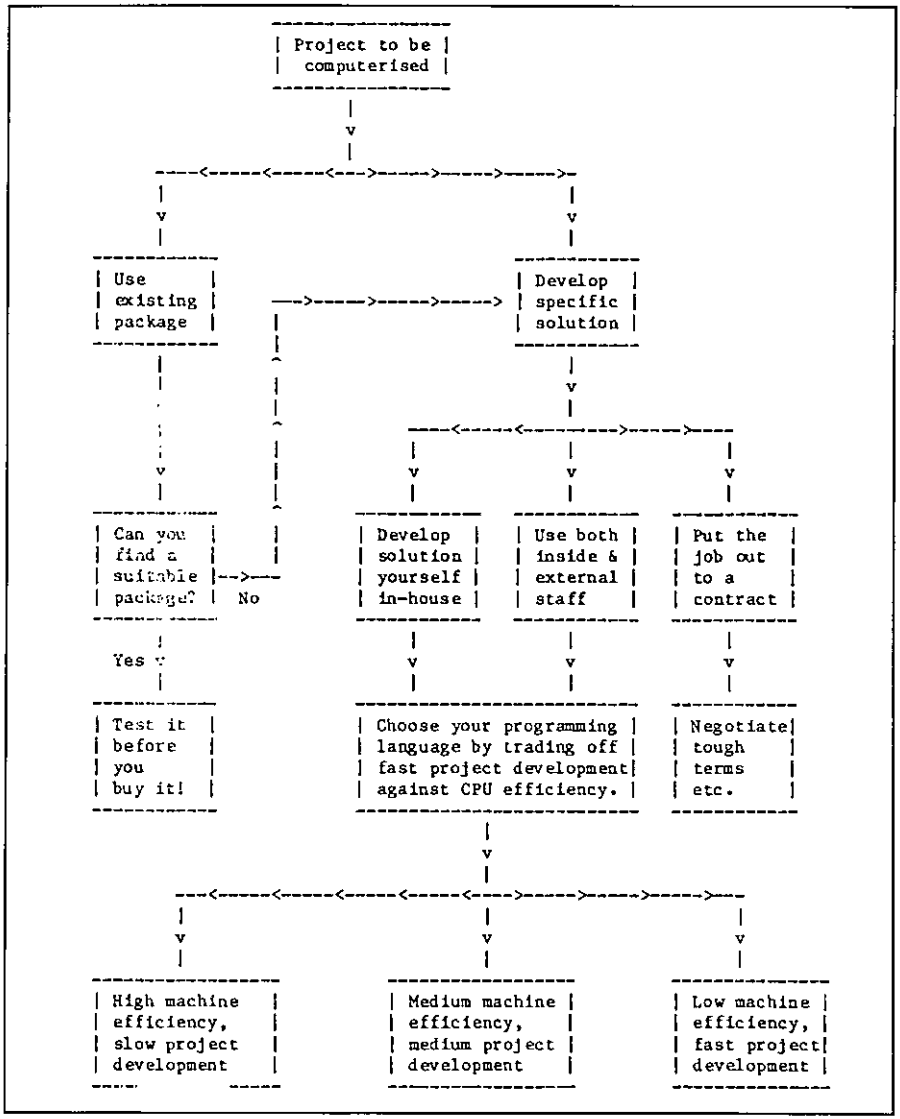
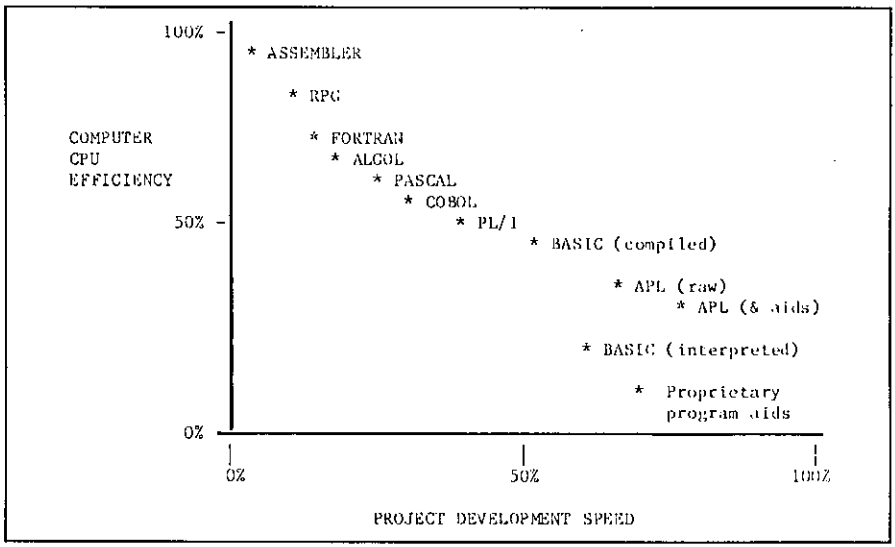


Fig.2 — The Programming Language Trade-off



A mix of in-house and external staff can be a very useful compromise. Instead of using the external people to write the whole system, you employ them to design the "architecture" of the computer suite and to train your own team on how to write the programs themselves. That way you don't get a large bill for many man-months of project development. You also find that your own people can maintain the system, make modifications and add new features to the system without incurring any more costs.

Efficiency vs. Flexibility

Now you have to choose what kind of programming language to use. Your choice may of course be restricted by what's on offer on the hardware that is to be used — but you should still take a decision based on an "ideal" set of resources at this stage. If you have to compromise, at least make sure you realise it. If instead you are expecting to acquire hardware specifically for this project, then your choice is completely unconstrained and you should certainly choose the programming language before you choose the computer.

There are really two kinds of programming language. There are Type A languages that were developed for the benefit of computers, and Type B languages that were developed for the benefit of people. Programs that are written in Type A languages take forever to develop, but once they're finished the computer eats them for breakfast. Programs written in Type B languages get finished nice and quickly, and then give the computer a hell of a job to run them within a given space of memory, with a given allocation of CPU time and so on.

Figure 2 illustrates this proposition. The scales I use and the positioning of the languages on them are based on my own empirical experiences, not on any glossy PhD papers; you may find that people dispute the exact placing of the different languages, but most of them accept the existence of the tradeoff.

You now need to decide what's important to you. Does it matter to your that your computers are used as efficiently as possible? That you don't waste any more CPU (central processor unit) cycles than you need in doing each job? That a machine of a given size can run the maximum number of simultaneous users? Or do you think that the goal of fast project development and

business relevance is more important than these things? Are you primarily concerned that the job is finished on time, within budget, and that the response is fast enough for the user's needs? Will you lose sleep worrying about wasted CPU cycles? Because one thing's for sure; you won't get 100% efficiency and 100% flexibility in the same language.

The Inherited Philosophy

I shall leave you to take that decision by yourself. However, you ought to be aware of the unconscious bias that almost all computer people have towards machine efficiency. Machine efficiency traditionally ranks next after cleanliness in its proximity to Godliness, and there's a good reason. Until the late 1970's computers were correctly viewed as large, expensive lumps of machinery whose utilisation had to be maximised. If you are planning to use a large central computer for your next job, you will undoubtedly discover that this continues to be true. The Data Processing manager has a large chart up on the wall which indicates % utilisation; everyone gets very relieved so long as it stays near 100%. If you want to be popular, try asking him (or her) the following question:

"What % utilisation did you get on your ball-point pen last week?"

If you survive the response, remind him that when the ball-point pen was invented (for pilots flying at altitudes where fountain pens would not operate) it cost about thirty pounds (a lot of money in 1932) owing to the precision low-volume manufacture of the tiny ball-bearing in the nib. Anyone who expected to use that particular resource made sure that their ball-point pen utilisation factor was as high as possible; pens were kept in a central locked store and withdrawn under signature. However, after some years a revolution took place in which mass production techniques drove down the price of the pen to a few pence. A few diehards were still left muttering about inefficient usage of the central resource.

The point is that it now makes sense to regard the provision of extra processing power as to all intents and purposes free. If that is not the case for your own company's data processing installation, that isn't a reason for failing to adopt user-efficient languages like APL: it's a reason for changing your data processing installation.

Given the advent of the microprocessor, the arguments in favour of user-efficient but machine-inefficient languages for most commercial applications have become virtually insuperable. If you don't agree, ask yourself whether you're arguing about the principle or merely the timing. I think you'll find that:

"The issue is when, not whether."

Depending on how much change you like to embrace at any one time, you may find that by pursuing this line of reasoning you end up with some rather interesting conclusions. Some of my own are summarised in Figure 3. I shan't comment on them here: let's leave them as food for thought.

Figure 3: Ten New Rules for Computer Projects

1. Systems cannot be specified until they have been implemented.
2. No single program should take longer than a day to write.
3. There is no such thing as the end of a project.
4. If the requirement does not change, the system is not being used.
5. All systems should be physically developed in the users' office.
6. The user is in charge of the computer project.
7. Lineprinters are forbidden.
8. Fields in files should change as frequently as records.
9. Databases were invented to employ system analysts.
10. If it moves, it will break this year. If it doesn't — next year.

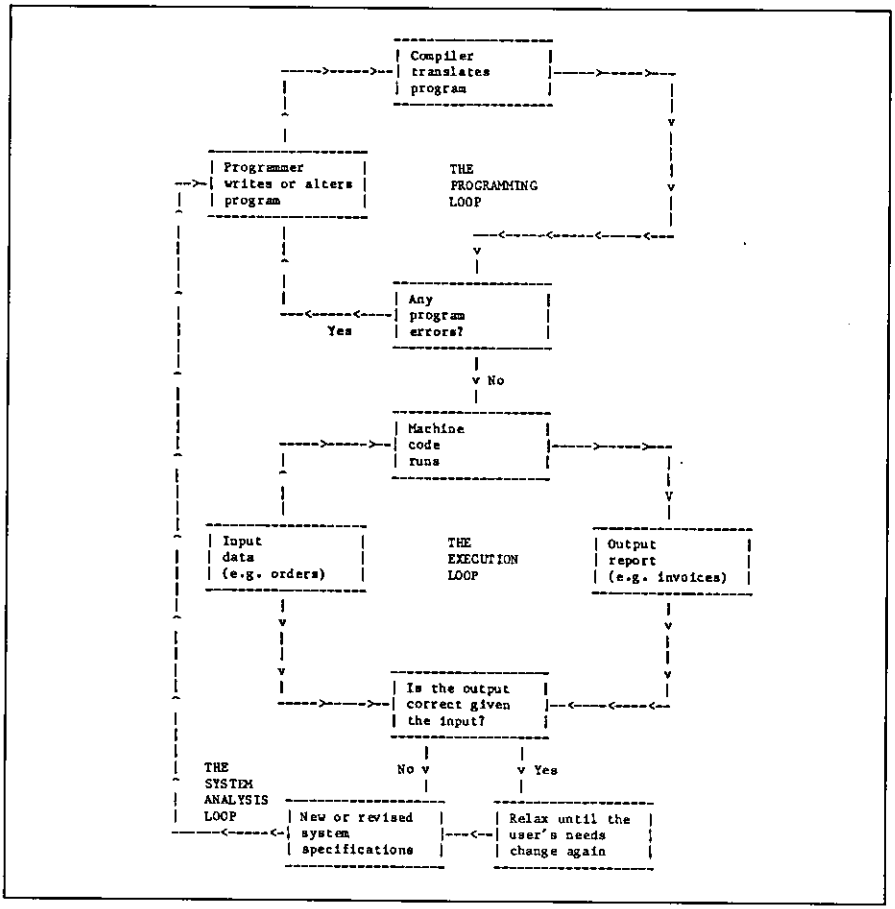
The Three Computing Loops

In Figure 4 I have tried to set out the main steps that occur when a computer application is generated using a language organised by a "compiler". Compilers expect to be given the whole program (or the main module anyway) at one time, which they turn into machine code (binary digits) in its entirety. This process may identify programming errors, which will be notified to the programmer, who has to go back to the program, find the problem, correct it and resubmit the program to the compiler. Depending on the size of the program, compilation may take anything from a minute to an hour or more. Any change at all in the program — even one mis-spelling on an output report — requires a complete re-compilation. I call this process "The Programming Loop".

Once the computer confirms that the machine code version contains no errors of a "grammatical" nature, it is given the input data and it produces whatever output report it is designed to create. This process may identify errors in the input data, which I haven't bothered to show on the chart since the solution is simply to modify the input data and run the job again. I call this stage "The Execution Loop".

The next thing that's probable is that the report isn't quite the right format, or the calculations aren't quite what was intended — in other words, there's no error in the programming, but there's a mismatch between the program specification and the user's current needs. These needs may themselves have changed as a result of seeing the output. That doesn't imply wickedness or sloppy thinking — it's a perfectly reasonable and sensible kind of learning process. So the system analyst revises the specification and submits a new request to the programmer. I call this third activity "The System Analysis Loop".

Fig.4: The Programming, Execution and System Analysis Loops



It turns out that languages exhibit intrinsically different efficiency in these three loops. Traditional languages typically maximise efficiency in the Execution Loop. However, the price that one pays for that extreme is that the Programming and System Analysis loops take a long time. Historically this didn't matter very much, because programmers' time was cheaper than machine time, and because users' requirements didn't change very often.

By contrast, the new generation of user-oriented languages such as some dialects of interpretive BASIC, some proprietary programmer productivity aids, and APL itself (particularly when enhanced with development utilities) believes that the Execution Loop is relatively unimportant, whereas the other two loops are vital. Consequently such languages are usually organised by "interpreters" rather than "compilers". Interpreters merge the process of compilation with that

of execution. The advantage is that errors can be much more precisely identified and far more quickly corrected, and that minor modifications can be introduced almost instantly without having to recompile the whole program.

The disadvantage is that in most cases the result is less efficient for the computer. In many commercial applications, this kind of efficiency loss means a response in one-tenth of a second instead of a response in one-hundredth. How do you feel about that?

Actually, APL happens to have a rather crafty way of cheating in this respect and producing results via interpretation which surprisingly often match or even surpass compiled program response. This is because a large number of frequently encountered cases are already "hard wired" into the APL interpreter on a hand-coded basis, which is generally more efficient than the result of a compilation. However, it looks too suspicious if I start trying to win all the arguments for APL, so let's agree to concede this one. It really doesn't matter very much anyway.

Figure 5: The APL Workspace

This workspace is called FRED

PROFIT
123 456 235 246 334 234

SALES
2345 4456 3456 3456 3457 2233
1353 5676 2356 5677 3356 6788
3456 3356 6434 2443 3454 2356

PRODUCTS
spangleworp Mark 3
defibrillicator
gudgeon sprocket

PROGRAM1
[1] Do this...
[2] Now do that...
[3] Do PROGRAM2
[4] Do the other...

PROGRAM2
[1] Try this...
[2]Try PROGRAM3
[3]Try again...

PROGRAM3
[1] Last chance...
[2] Give up.

It has variables PROFIT, SALES and PRODUCTS in it, and programs called PROGRAM1, PROGRAM2 and PROGRAM3 (they could be called anything).

Oops — it's lunchtime, so: SAVE FRED saves all the programs and all the data on the disc, ready for me to start this afternoon with LOAD FRED to get them back again.

Hmmm — that program called SMARTONE that Joe developed the other day could be handy here. Let's borrow it. I have permission to read his workspaces? Great: COPY JOE SMARTONE gets me a copy (leaving the original with Joe).

Ahhh — I'd like to browse through my data, looking at elements of that table called SALES as I go and changing some entries. No need to write a program: APL desk calculator mode lets me do all that as standard.

I can use all the built-in APL operations to experiment with my data in desk calculator mode. When I've got the sequence right, I can type them all in as lines of a program.

Subroutines? Any program in APL can be called by any other program without prior arrangement. You don't have one big program in an APL workspace; you have lots of small ones with links between them.

The real advantages of APL

In most of the articles about APL which I have seen, the author starts rhapsodising at this point about all the amazing symbols that exist and all the wonderful things that you can do with them. Doing wonderful things is fine, but the use of special symbols apparently dissuades as many users as it encourages, not least because a special screen or printer is required. Somewhat belatedly the APL community is waking up to this fact, and several projects are in hand to enable effective use of APL from standard non-APL screens (ASCII or EBCDIC etc.). So please don't regard these non-standard symbols as an inherent part of the APL philosophy.

Figure 5 shows one of the APL advantages: the workspace. Until you've used it, you simply can't imagine how flexible it can be. A short summary of the real benefits of APL runs as follows:

a. The Workspace

Everything is done in the APL workspace. Even with micros, this is at least 200,000 bytes (characters) or more. Your data is independent of your programs and it can be generated and edited without using a program at all. You can save them, load them, copy all or part of them, and you develop all your programs in them.

b. Data variables

These can be text or numeric and can have almost any "dimension". Zero-dimensional variables contain a single number or character. One-dimensional variables are lists or "vectors". Two-dimensional variables are tables or "matrices", and so on, up to 8 or more dimensions if you want them. See Figure 6 for details.

c. Built-in operations

These are numerous. All the standard arithmetical functions, all the algebraic and trigonometric ones you're ever likely to need (plus a few more), all the comparatives (greater than etc.), the logicals (and, or etc.)... There's a built in numeric (and usually character)sort; there's text and numeric searching, finding minima and maxima, random number generation, array indexing and manipulation, numeric formatting, number base changes, even matrix inversion. And the great advantage is that nearly all these built-in operations apply directly to arrays of data without the need to write loops.

d. Programs

APL programs have a unique architecture; the nearest equivalent is a little-used language called FORTH. In APL you very rarely write a long program. A given project is usually implemented by a workspace consisting or twenty or more programs. Each does some small part of the job, rather like a subroutine in other languages. But the subroutines in APL are completely freestanding and have their own identity. So a very well organised control structure emerges in APL by arranging for one program to call another to do some appropriate job. Phrases like IF...THEN...ELSE or DO... WHILE are hardly ever needed in APL; partly because loops themselves are not needed for operations on arrays, and partly because the concept of having many programs operational at one time makes these phrases unnecessary.

Figure 6: Examples of APL Variables

(To avoid confusing the reader and the typesetter with the traditional APL symbols, I have used a close ASCII equivalent. Various alternate forms of ASCII-type syntax for APL expressions are currently on trial by different APL suppliers and undoubtedly a standard will emerge to bring the benefits of APL to those who do not wish to modify existing hardware or acquire new terminals.)

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

     fred←3 5 SIZE LIST 15

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

     fred [3;]
11 12 13 14 15

     fred [;2]
2 7 12

     fred [3;2]
12

     fred [3;2]←999

     fred [;1]←fred [;1] - 1

     fred
 0   2  3  4  5
 5   7  8  9 10
10 999 13 14 15

     joe
a p p l e s
O R A N G E
t o m a t o

     joe [;3]
pAm

     joe [2;4 5]←'CL'

     joe
a p p l e s
O R A C L E
t o m a t o
```

Stumbling on APL via BASIC

Everyone knows the boring old joke about the Irishman who, when asked how to get from A to B, replied "I wouldn't start from here if I were you". It's a bit like that if you start considering APL from a background of knowing any other language. I'm frequently asked how long it takes to master APL if you're being trained by, for example, a computer-based APL self-teaching course. My response is always:

"Do you already know another computer language?"
"Yes — BASIC and a bit of FORTRAN."
"About a fortnight then."
"And if I didn't"
"Oh, not more than a week."

Part of the battle if you're already computer numerate it to:

FORGET ABOUT:	AND START USING:
* Loops	* Array syntax
* Big Programs	* Small program modules
* DATA statements	* Independent variables
* Control structures	* Program nesting
* Operations on single numbers	* Operations on arrays
* File design	* Workspace operations
* Specifying the whole system	* Prototyping techniques
* Sequential processing	* Parallel processing
* Code names for data	* English descriptive names
* Accessing data from a program	* Desk-calculator access
* Integers vs. floating point	* Transparent data conversion
* Man-months	* Man-days

It gets easier after a little practice, and you'll never look back.

Files and Inverted Data Structures

In most languages, file design is a constant headache. You don't seem to be able to do anything much without sitting down and specifying a whole lot of boring file formats; and once they're fixed, they're bastards to change. In APL you often find that you can implement a whole project without using a file at all. This is because the data that you specify starts by living in memory, and if there's enough room and if nobody else wants to update it simultaneously, you might just as well leave it there. When you load the system each morning, you load all the data as well.

However, files certainly play their part in APL. You can create files which look just like the sort of files you get in COBOL or BASIC if you want to — the facilities are there for that sort of thing. But APL makes it delightfully easy to standardise on a so-called "inverted" (I would rather call it "transposed") format for your files, as illustrated in Figure 5. The advantages are enormous; the example of Figure 7 should make that clear. APL can effortlessly perform the kind of job that in other languages you'd need a relational database or a context-addressable memory for. In fact in some ways APL already is a relational database.

Figure 7: File Structures in APL

CONVENTIONAL FILE STRUCTURE

0001	FOSTER	JOHN F.	32	ACCOUNTS	1981	16500
0002	CARTER	ERIC P.	37	SALES	1976	18900
0003	BLOGGS	CHARLES	45	H. OFFICE	1965	25000
. . . . .						
. . . . .						
. . . . .						
2000	TOMLINS	JULIA H.	25	RESEARCH	1982	19000

Files are organised by rows. Each row contains all the data on a particular person. Access is sequential or by row number or 'key' (ie. a nominated attribute of a person, such as Name). Requests for information such as:

"Average age where salary above 18000 and department is accounts"

require a sequential scan through the entire file with as many disc accesses as there are rows. This is measured in minutes even for files with only 2000 entries.

TYPICAL EQUIVALENT APL FILE STRUCTURE

1	2	3	4	5	6
FOSTER	JOHN F.	32	ACCOUNTS	1981	16500
CARTER	ERIC P.	37	SALES	1976	18900
BLOGGS	CHARLES	45	H. OFFICE	1965	25000
. . . . .	. . . . .	. .	. . . . .	. . . . .	. . . . .
. . . . .	. . . . .	. .	. . . . .	. . . . .	. . . . .
. . . . .	. . . . .	. .	. . . . .	. . . . .	. . . . .
TOMLINS	JULIA H.	25	RESEARCH	1982	19000

Files are organised by columns. Each column contains all the data on a particular attribute. Access is by column number; the whole column is read in with a single file access, and it can be scanned using APL array operations to find matches, select entries based on some criterion and so on. A question like:

"Average age where salary above 18000 and department is accounts"

requires three files accesses, for columns 3, 4 and 6, and the question will typically be answered in a few seconds.

APL Program Development Aids

APL is good, but it's an even better idea to employ pre-written programs in your own APL applications. There are two main types of development aid or "utility program" as they are frequently called.

There are utilities that you could write yourself if you had the time, and there are fast machine-coded added features which you probably couldn't unless you happen to be an expert in the Assembler dialect used on your hardware. These last are usually called "auxiliary processors", although they refer to software, not to hardware. Typical useful utility library items include:

* General purpose program development aids, data input routines etc.
* Full-screen program or variable editors
* Full-screen forms design and data entry systems
* Communication links to other computers or non-APL file formats
* Screen handling primitives for device-independent control
* Automatic documentation facilities
* Graphics routines for colour screens and flatbed plotters
* Specialised file access routines for specific purposes
* CAI (computer-aided-instruction) APL self-teaching programs
* Spreadsheet "front-end" modules for use under APL control
* Specialised interfaces for on-line inter-user communications
* Printer handling features such as spool queue management

APL suppliers differ in their ability to supply this kind of library software. Furthermore the APL interpreters themselves, although adhering for the most part to a common language 'core' defined by IBM, differ considerably in terms of the number of optional enhancements that they provide. Of course, most APL users don't require every APL facility that's ever been thought of, but it's surprising how quickly a user matures to a point where a feature such as programmed error control, for example, becomes mandatory rather than desirable. The moral is to shop around!

Conclusions

APL programmers take for granted the kind of capability which leaves most other languages gasping. Not surprisingly, very few people who know APL ever become disillusioned and turn to another language. Until recently, however, APL was somewhat inaccessible to many potential users, either because of the need for a large mainframe or because of an initial reaction against its special character set. The first problem has been solved since 1981 with 16-bit microcomputer versions of APL; the second is on its way. So you really have no more excuses. Good luck and good programming!

Steps Towards a Better Basic - part 1

by Anthony Camacho

When you want to know the value of a variable in BASIC you have to type ?VAR or P.VAR or even PRINT VAR. Just typing VAR has no significance. If it were given the same significance as PRINT VAR nothing would be lost and a good many key depressions saved. To find out the value of an array is even worse; you have to write a loop. There could scarcely be any loss if the same were done for arrays. Millions of BASIC programmers would be grateful if they could type ARRAY and have it displayed for them. Most of the "programmers' toolkits" miss this opportunity to be really helpful; I have one that responds to DUMP with a display of all the single variables and their values but totally ignores the arrays.

This reluctance to deal with more than a single variable is odd because BASIC already deals with some multiple variables as if they were single. The string NAME$ may be ten or a hundred characters long — a thousand in some versions of BASIC — yet it can all be displayed by PRINT NAME$. Some of the loops in BASIC could be cut out by making this method of handling multiple variables available for numbers as well as letters. In that case NOS could contain 1 2 3 4 5 (or a list of works order numbers or part numbers).

Strings are only moved about, compared and printed; arithmetic on multiple variables brings complications. If one array holds stock quantity and another holds the prices for the corresponding items, the instruction LET VALUE = QUANTITY * PRICE would create the new array called VALUE. Thus it is obviously useful to be able to operate on each variable in one array with the corresponding element in another array of the same shape, and to produce a third array of the same shape as the result. But there are occasions when a whole array needs to be multiplied by a single factor: exchange rates, discounts and inflation rates are factors that come to mind. There is no reason why DISPRICE = PRICE * .90 could not create the array DISPRICE which is the price after 10% discount.

Such facilities offer the opportunity to create some more functions, such as would add the rows of an array or the columns or even all the items. It could be useful to convert an array of another shape (for convenience in printing it, if nothing else). There could also be a function to take each item of one array with each item of another. It could be called EWE (for Each With Each) and be followed by the operation needed in brackets, so NOS EWE(*) NOS would give:-

1	2	3	4	5
2	4	6	8	10
3	6	9	12	15
4	8	12	16	20
5	10	15	20	25

My excellent cheap calculator changes its display for very small and very large numbers; they are shown in scientific notation. As in this improved version of BASIC, character and numeric variables can both hold single or multiple variables, there will no longer be any need to decide in advance which type of value each variable is to hold. VAR could now hold either a string or a number, and the interpreter could follow the example of my calculator and distinguish between numbers of different types; it will adjust the way the number is held from integer to floating point and back according to its value.

Of course this implies that all arithmetic will be done to some standard accuracy, because the interpreter would no longer know from the variable name whether a variable was wanted as an integer or single or double precision number, so would have to do everything in double precision.

* *Reprinted by kind permission of DATALINK magazine*

Very few calculations would be appreciably slowed by this and all would be speeded up to a small extent by having only one method of handling. As computers get faster and cheaper the loss of speed will be less and less noticeable. And anyway if speed is important, the program should not be run under an interpreter but should be compiled.

String handling in BASIC demonstrates that in an interactive language there is no need to DIMension multiple variables. It is quite practical to assign them dynamically, but if that were done there would be a need for a function such as LEN (which returns the length of a string) for arrays. If the arrays were to be limited to one dimension then LEN would still be adequate, but it is often convenient to have tables of numbers (or lists of words) and it would be better to introduce a new function which reported all the dimensions of an array; it could be called SHAPE. The only reason this was impossible before was that the result could have had one or several numbers in it. Now that a variable will be able to hold one or several values there is no difficulty. So, in the example above SHAPE NOS would be 5 because NOS contains five numbers. SHAPE NOS EWE(*) NOS would be 5 5. Also a way of extracting one or a group of the numbers will be needed. For strings BASIC has MID$(NAMES$,3,2,) to extract the third and fourth letter. In Sinclair BASIC, NAMES$(3 TO 4) does the same job better so NOS[3 4] could do the equivalent for the numbers, and pick out the third and fourth items. And (NOS EWE(*) NOS)[3 4;4 5] would pick out

```
12 15
16 20
```

(each dimension separated from the next by a semicolon, taking lines first and columns second).

One possible objection that might be raised to this is that there would be many more reserved words. Even with BASIC as it is now there are occasions when a reserved word is inadvertently used as a variable name. If more reserved words are to be added steps should be taken to avoid confusion. Variables could be kept in lower case for example. This is already common practice with the Sinclair Spectrum and the BBC. It makes programs less tiring to read too. Originally BASIC was used on teleprinters without lower case but as even cheap microcomputers have lower case now there is little point is keeping to upper case only. It might be sensible to use the capital X as a reserved word for multiplication, for those who find the asterisk annoying.

Now for the surprise: this description, in all essentials, fits a language in current use on hundreds of microcomputers and mainframes. The language has a great many more attractive features which will be explored in further articles. It has a reputaton for being difficult because of the peculiar characters it uses instead of reserved words, and it cannot become widely popular while keyboards and displays and printers have to be specially adapted to use these characters. It has most of the virtues of such languages as LISP and LOGO and FORTH and a richer collection of functions than any other language. It should be more widely known. Its name is APL.

CASE STUDIES: AN INVITATION

by Adrian Smith

Let me begin by stating my personal objectives as editor of this section of the Journal:

- to include in each issue a clear and concise account of a self-contained (but not necessarily simple) APL system, or of some specific aspect of a system (such as the use of colour) which readers can appreciate 'out of context'.

- to collect such accounts from as wide a range of areas as I possibly can.

- to stress ideas and methodology rather than technological cleverness. I anticipate a rather low percentage of actual APL code.

- to include as much illustrative material (if possible photographs) as company sensitivity allows.

In general I am happy to play the journalist and do the actual writing bit, but if any reader has an account which roughly fits the above outline, please do not hesitate to send it in. My standards for such copy are set very low indeed viz:

- typescript preferred, but legible handwriting quite acceptable if the content is right.

- diagrams decently drawn in black ink (assuming you want me to paste them straight in to the final version).

- function listings as good as you can get them. Please use six lines per inch rather then eight on IBM dot-matrix printers, and put a new ribbon in!

- colour is not (as yet) reproducible, so make sure any graphics are acceptable in monochrome.

In short if it's sufficiently interesting I'll accept almost anything — don't let the effort of 'writing up' put you off. Since there were no contributions for this first issue I have taken the liberty of contributing an article myself. I found myself more than willing to accept it.

Matchmakers: a Case Study in Simulation

by Adrian Smith

Problem Description

Matchmakers are a boxed chocolate produce from Rowntree Mackintosh. Following manufacture, the lines of Matchmakes are scooped into trays, which are inserted into sleeves before final packing. The sleeving process, currently done by hand, is to be mechanized. The running speed and breakdown pattern of both traying and sleeving machines can be estimated from past experience.

Question: how large a buffer is needed between these machines so that the plant can be kept running when the sleever breaks down?

Basic engineering reason (like the height of the ceiling) restrict the choice to buffers of between 10 and 20 minutes, with the cost increasing roughly in proportion.

Hypothesis

With a bit of luck both Traying and Sleeving will stop at random, and the length of stops will show a nice exponential pattern. If so the logic is easy, because in any given time interval (say 1 minute) there will be a constant probability of stopping (PROBSTOP) and having stopped a constant probability of restarting (PROBSTART). All that has to be done is to determine the frequency and half-life of stops, and we can then generate endless random days and see the way different buffers fill and empty over time.

Snag

By an unfortunate coincidence the foreman had only just cleared out his cupboard, evicting in the process all last year's records! There was thus a short delay for...

Data Gathering/Valuation

Over the next few weeks the machine logs were collated for several comparable pieces of plant. To the ill-concealed delight of the experimenter the emerging distributions looked like Figures 1 and 2.

Fig. 1: Analysis of Stoppage Time on Traying Unit

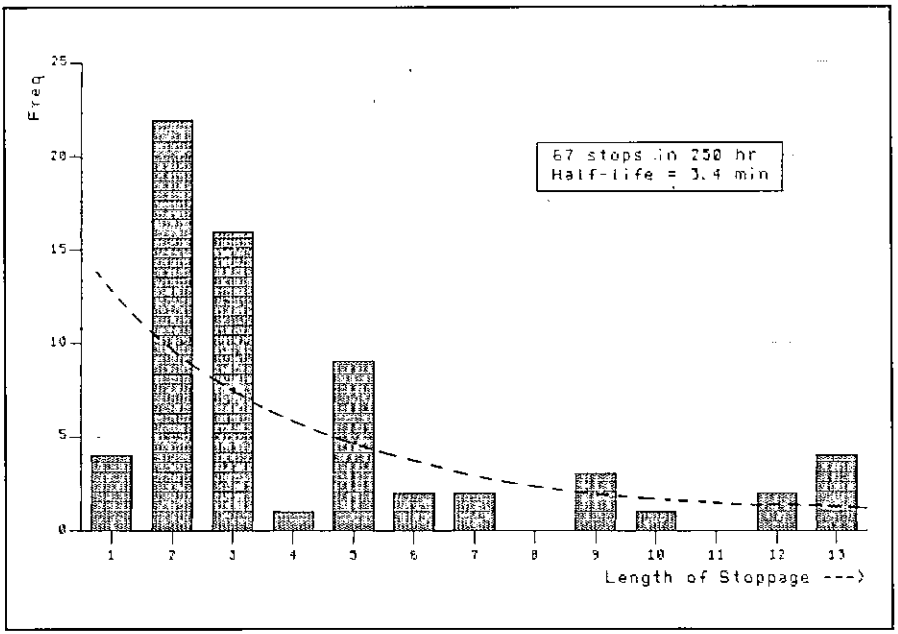


Fig.2: Analysis of Stoppage Time on Sleeving Unit

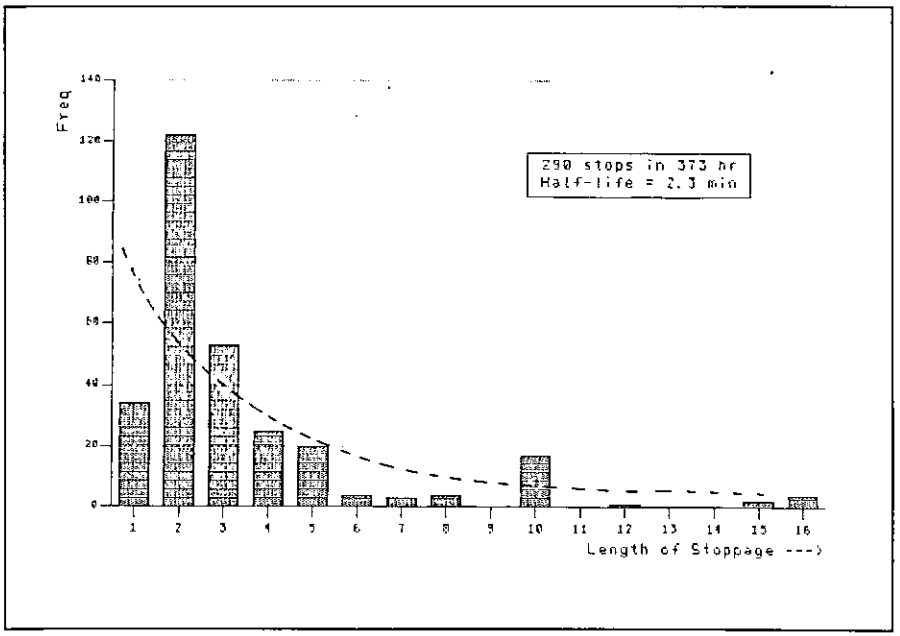
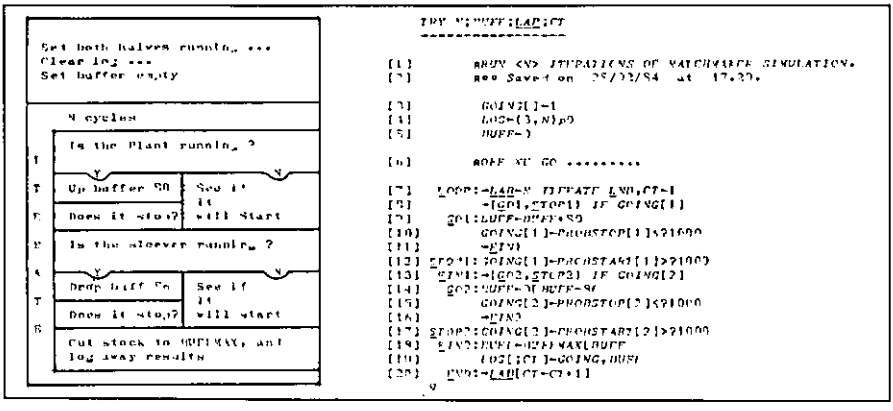


Fig.3: Simulation Logic



Taken together with the total number of stops in the total logged running time the graphs give:

PROBSTOP [T S]←4.5 13 (per thousand mins running)
PROBSTART [T S]←180 260

Assuming an arbitrary upper limit (BUFFMAX←1200) to the buffer, the logic for the simulation now looks like Figure 3.

The results were analysed with a simple COUNT routine (TRY 480 was typical for an 8-hour day); each run giving a report like:

Stops on traying : 5 3 3 1 7 3
Stops on Sleever : 7 1 3
Buffer full for : 10 min out of 480

Enough sample days were run to check that the generated patterns of stoppage were indistinguishable from the real thing, and several 'typical' days were graphed to ensure nothing odd was going on in the logic. The results are depicted in Figures 4 and 5.

Fig.4: Matchmaker Simulation: 4.5 13 180 260

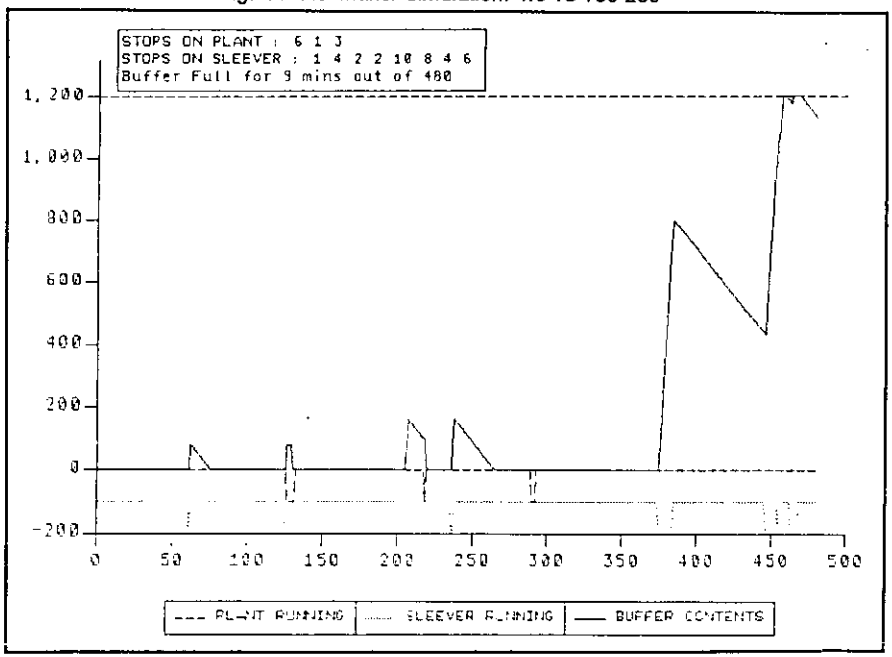
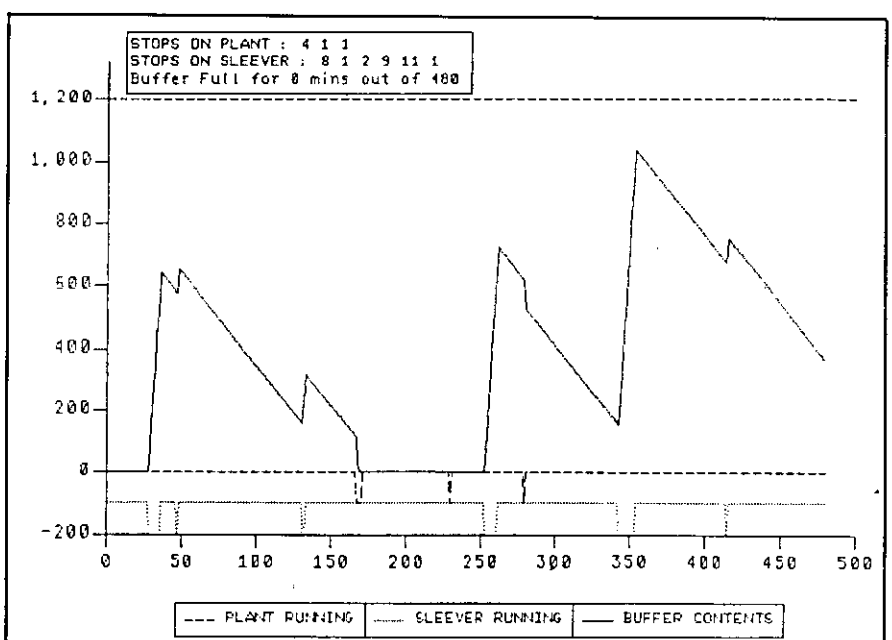


Fig.5: Matchmaker Simulation: 4.5 13 180 260



As soon as all concerned were happy with the pictures, the project moved to its final stage .....

Production Run/Problem Resolution

Out of consideration for other computer users this was done 'out of hours'. Several runs of 28 days were done at each possible buffer size, and the number of days affected by at least one 'buffer full' was plotted. Also shown is the percentage of production-time lost in the period (Figures 6 and 7).

At this point it becomes clear that the problem as originally formulated does not have a satisfactory answer; however big the buffer there will always be some days affected. What we can see is the possibility of a trade-off between the cost of lost production (which does fall away quite sharply) and the cost of installing buffers of various sizes. I'll spare you the details of the economics, suffice it to say that a decision was rapidly reached which pleased all the interested parties.

Fig.6: Analysis of Days affected per Month on Traying Unit

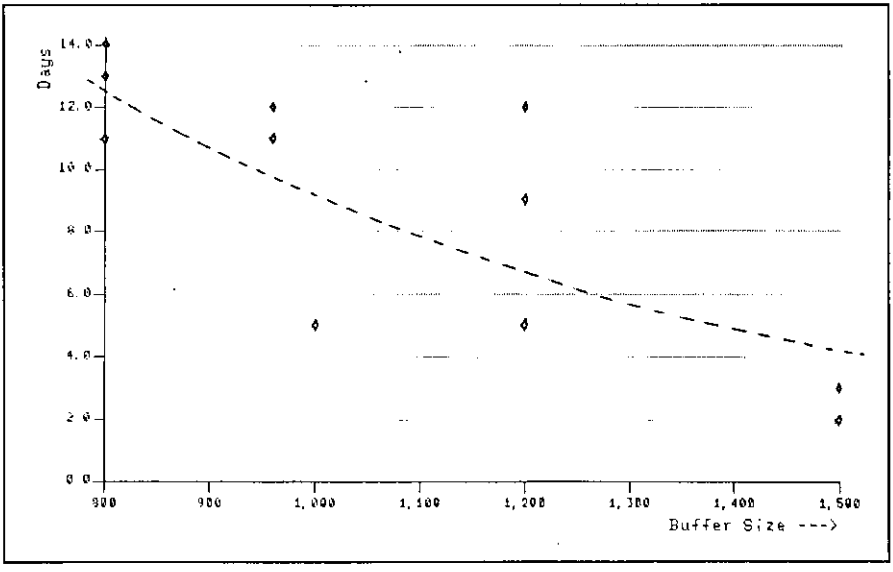
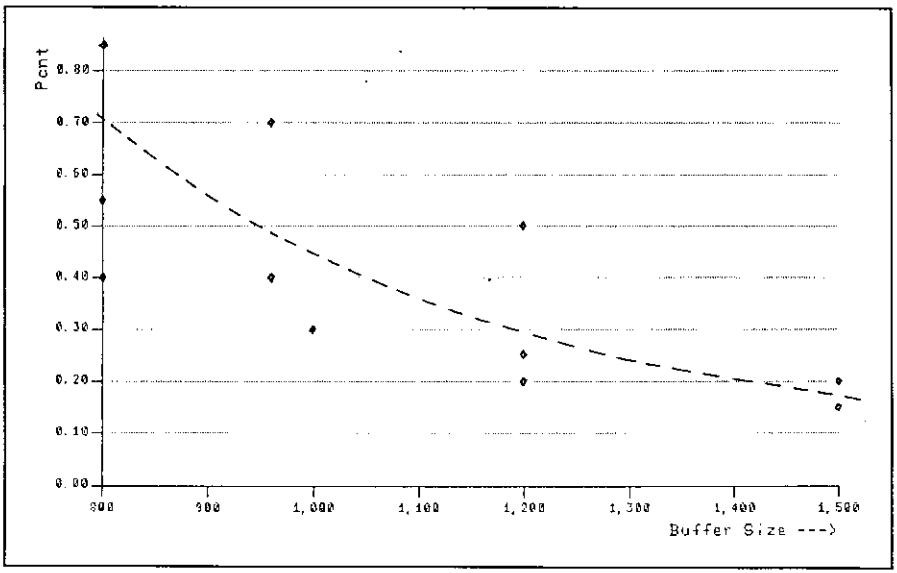


Fig.7: Analysis of Percentage of Production Lost due to Full Buffer



Comment

APL is not an ideal tool for simulation — a run of 28 days of 480 minutes (i.e. 13440 iterations of the basic model) cranked up around 20 sec of 3083 CPU time, and several such runs were needed. On the other hand the coding for the whole exercise (including the calls to PGF for the graphics) took just over an hour, and it all ran first shot (once the typos had screened themselves out with SYNTAX ERRORS).

Some other points of interest are: the use of block diagrams to sketch out the logic (yes I did draw the pictures first!); the veritable shoal of helpful utilities (IF and ITERATE are ones I use constantly); and the power of graphics as an analytical tool. I dare say any competent mathematician could have got to the same answers as I did without any computing at all, but try selling that kind of answer to a bunch of engineers and production managers! The use of pictures was important in validating the model, and thus giving credibility to the answers.

Metapraxis Ltd.

Metapraxis is a consultancy company involved in the creation of advanced computer-based management control systems. We are currently embarking on a major new project to develop and market a range of control centre software. Some of this software involves the development of IKBS modules (expert systems). Substantial City and Government backing for this project has been arranged and the development phase will commence shortly.

We are now interested in hearing from individuals with outstanding track records in one or more of the following areas:

* *Programming (particularly Assembler, APL, PROLOG or LISP)*
* *Financial control in the large company environment*
* *Graphic design and publicity expertise*
* *Marketing skills for new product launch*
* *Sales of high-level products to senior corporate managers*
* *Industry analyst skills in a major commercial sector*

There are also one or two vacancies for exceptional graduates with little or no work experience. Candidates should be prepared to assimilate many unfamiliar concepts in a very short space of time and to become an authority in at least one area of the company's activities. Applicants should write in confidence with career details (indicating which job is of interest) to:

Metapraxis Ltd., 26 Barham Road, London SW20 0ET

THE PAPER THEY DARED NOT PRINT

by Anonymous APL84 Referees

It has now become a noble tradition to submit to the Programme Chairperson of each annual international APL conference a paper so scurrilous, so extreme and so generally unacademic that it cannot possibly be accepted. For the 1983 APL Conference the accolade of "The Paper They Dared Not Print" was captured (no contest) by an article entitled "The Coming Revolution in APL", which proposed major changes to APL for it to become more widely used as a general purpose language. Although the paper was rejected, the APL 83 organisers were kind enough to give it space at a plenary session; its bootleg presentation to an audience of over 700 was the occasion of some light relief as a contrast to what Ron Fuss of IBM has christened the "dot dot comma slash slash star" brigade. Avid seekers of the alternative truth about APL can find that article in Issue 6 of "MicroAPL News", a trade journal from one of our sustaining members.

"A paper so scurrilous and extreme that it cannot be accepted"

To keep the Finns on their toes this year, two papers were submitted to the Programme Chairperson that stood some reasonable chance of rejection and true to form, rejected they both were. For those readers who don't know about the process, APL conferences papers are circulated to three referees, on the basis of whose comments a paper is accepted or not. A good way for a Chairperson to keep a potentially embarrassing paper off the list is to send it to referees who are guaranteed to have apoplexy at anything immoderate. In this respect the choice of referees for these papers represented an outstanding success for Mr. Kallin (always assuming he exists: see earlier International News section).

But unlike the formal APL proceedings, VECTOR is a journal dedicated to printing the truth: "We name the guilty men" and so on. Although in this case we can't quite do that since the referees are anonymous, we bring you the next best thing; we let you share their indignation. And we print the papers. One of them you've already seen if your approach to VECTOR is sequential rather than random access: "Why APL?: A Non-Technical Introduction". The other is called "FGL: Fifth Generation Language" and it comes next. The fact that both were created by your Editor should only whet your appetite for some savage letters for us to print in the next issue.

"Try to avoid the word 'bastard' as a description of non-APL files."

Papers were assessed on five criteria. The "Why APL" paper gets slated by one referee as duplicating existing work: "Would be better published in a popular press microcomputer magazine". Another referee says: many attendees would be interested in it, there are some new ideas, the length is appropriate, the technical quality is high, and the use of English is "outstanding". However: "I'd prefer not using the word 'bastard' in reference to the difficulty of changing a file structure". Ah yes, but has he ever tried? The diagram called the "Programming Language Tradeoff" which is explicity documented in the article as based only on empirical observation receives the damning comment "Is this made-up data? Frankly I don't believe it", whereas the same referee surprisingly annotates the "Ten New Rules" with the pithy endorsement "This one is OK". Well, thank goodness for such technical praise.

"BASIC has nothing to do with science: did you mean science fiction?"

However, it's only when we get to "FGL: Fifth Generation Language" that the flak really starts flying. On the matter of audience appeal, we have one referee (A) who thinks the paper would be of interest to everyone, another (B) who says "specialists only", and a third (C) who can't decide. On the thorny issue of originality, A says 'very original' and B agrees, while C ticks the box saying "duplicates existing work — specify", but he doesn't specify, so we shall never know. All the referees agree that the paper is much too long, so some of the tables have been omitted in this reprint to try to make amends. The technical quality comes in as variously high and low. Finally, the vexed question of the use of English scores one 'oustanding', one 'good' and one 'needs thorough editing'. As far as comments are concerned, there were many, including "Remove humorous reference to AIDS — in poor taste" and "BASIC has nothing to do with science: maybe you meant science fiction?"

This paper is also published with some reservations from the VECTOR editorial team as a whole. Controversy is expected. The common disclaimer that the opinions expressed in a paper do not necessarily correspond to those of the journal as a whole should be taken seriously. But we are after a little light relief amidst the serious cut and thrust of computer programming, and this next article is for those of you sitting in trains after a long day writing incomprehensible APL code at the office. By way of a footnote: BYTE politely rejected the article as "not of general interest to the readership", while the paper's Hofstadterian self-reference to its own acceptance at Helsinki was another of the many straws that broke the camel's back. So APL will no doubt continue to sail merrily along in its delightful little backwater while the rest of the world computes otherwise.

Your Editor happen to be an APL 84 referee too, but sheer terror at the possibility of printed reprisals always causes him to heap praise and unconditional recommendation on every article that is sent to him.

FGL : FIFTH GENERATION LANGUAGE

by Robert Bittlestone

Abstract: FGL (Fifth Generation Language) has been the world-wide norm for most computer programs since the late 1980's, when it replaced BASIC as the de-facto industry standard. Equally suitable for business, scientific and knowledge-based expert systems applications (which represent of course the main use of computers these days), FGL was first implemented in 1984 and quickly rose to prominence as a language of unmatched elegance, economy, and program development speed. Running on almost any computer and using only the industry standard ASCII character set, FGL nevertheless bears a striking structural resemblance to a little-known and now obsolete computer language called APL (A Programming Language). Students of the history of computing may be interested to see how FGL developed from its foundations in APL and how it was influenced by the success of BASIC.

* * * * *

Some of our older readers may have been taught BASIC at school in the early 1980's and it's worth remembering why that language became so popular around that time. In the mid-1970's the first practical implementations of von Neumann architecture computers on silicon microprocessors became available. The prevailing architecture at the time was 'bit-parallel'; the hardware quickly moved through 4-bit, 8-bit, 16-bit, 32-bit and by 1985, 64-bit CPU generations. By today's standards these 64-bit processors were of course primitive: the concept of today's highly parallel architectures composed of over 20 Giganodes of SHEEP (synapto-heuristic evolutionary expert processors) was of course a little over the horizon.

It's difficult to remember that in the 1970's, computer power was still finite and limited, and the random access memory (RAM) capacity for personal computer users was an incredibly confined maximum of 64k storage units, or "bytes" as they were called. Even by 1985 the average personal computer user had as little as 1024k bytes or a "megabyte" of space to play with. No small wonder, then, that the objectives of the languages that people were using then still included the quaint old ideas of:

- minimising the space used by the language interpreter/compiler
- maximising the efficiency of the machine code produced
- making it reasonably easy to create the interpreter/compiler

A project group at a location called Dartmouth (believed by some to be in the county of Devon in latter-day Britain, now part of New Thatcherland) developed BASIC in response to these goals. You can find out about BASIC's capabilities in any good reference book on the history of science; here we will describe its main shortcomings by the standards of the day.

Some BASIC Drawbacks

- Only one program in memory at any one time.
- All subroutines had to become a physical part of this program.
- Data was either stored in 'files' or was declared in the program.
- Data could not be inspected without a program.
- No array-handling features in the standard language.
- Virtually no standardised text or string-handling facility.
- A pathetic selection of arithmetic and other functions.
- Bizarre limitations on variable names.
- Inability to pass data to subroutines via parameters.
- No concept of 'workspace' for developing programs and data.
- No built-in screen and printer control standards.
- Fixed-length record-oriented file structures.
- No concept of personalised command languages.
- User-defined functions limited to one line in length.
- Maximum of two dimensions in an array.

The list goes on. By today's standards, BASIC can hardly be classed as a computer language; it's more like an ingenious extension to binary notation. However, BASIC did do one thing: it made computers available on microprocessors to millions of people who would otherwise have missed the opportunity to use them. Now some may say, of course, that it would have been a better thing for mankind if this had never happened. No Third Industrial Revolution, no gangs of neo-Luddites roaming the streets with their PROMsniffers, no AIDS riots (Artificial Intelligence Destroys Society), and no fatal cases of Syndrome 2480, the tragic mystery illness that's filling our hospital beds today with zombie-like patients suffering from an apparently irreversible degradation of vision into a series of rectangular cells. But the picture isn't all black. Without computers — and Fifth Generation Language in particular — we wouldn't have solved the world's energy crisis, stabilised the global weather map, put an end to Third World poverty, brought the national economy under perfect control, etc. So let's take a look at how FGL made it all possible. What follows isn't a formal definition of the language: you can find that in the 1986 edition of the proceedings of the FGL User Group Meeting.

FGL Concepts

There are only ten concepts in FGL. These are:

—	Names	The rules for composing an FGL object's name
—	Modes	The different ways in which FGL can be used
—	Workspaces	Where a user's data and programs are developed
—	Variables	Text or numeric, single or multi-dimensional data
—	Intrinsics	Built-in data manipulation facilities
—	Operators	A way to apply intrinsics more powerfully
—	Functions	Users' programs and subroutines
—	Commands	Instructions affecting the FGL environment
—	Extensions	Implementation-dependent additional facilities
—	Files	Optional external storage of variables or programs

FGL Names

Workspaces, variables and functions are assigned names by the user. These may be of any practical length and are drawn from the alphabet and the digits. Various reserved upper-case names are used by intrinsics. The first character of a name must be alphabetic. Users are encouraged to use lower-case for their names.

FGL Modes

You can only be doing one of three things at any particular time in FGL. In 'function execution mode' you are running a program that you or somebody else has written; in 'function editing mode' you are writing or modifying a program yourself; and in 'immediate execution mode' you are trying out FGL expressions by using the language as a kind of super-calculator. FGL's rich repertoire of built-in intrinsic operations allows you to do a great deal of work in immediate execution mode without even bothering to write a program. The data resulting from program runs are available for inspection at any time, including the contents of multi-dimensional arrays; they don't perish when the programs stop running.

FGL Workspaces

An FGL workspace is an area in the machine (historically RAM, now multi-processed in SHEEP) in which FGL programs and variables are created and in which they normally reside, both when they are in use and when they are stored away somewhere (historically spinning around on 'disc'! — now kept in BUBPAK, LASERCARD or CRAMPAK). An FGL workspace might typically contain some thirty or forty functions and perhaps ten or twenty variables. Some of the functions would

Table 1: FGL Intrinsics

Intrinsic	Monadic Case	Dyadic Case
+	Force display	Add
–	Prefix to negative number	Subtract
#	Sign (+1, 0 or –1)	Multiply
%	Reciprocal	Divide
MAX	Round up	Select maximum
MIN	Round down	Select minimum
|	Absolute value	Remainder
*	e to the power	raise to the power
LOG	log to the base e	log to any base
!	factorial	combination
TRIG	pi times	sin, cos, tan etc. (15 options)
LIST	list from 1 to ...	find position in a list
?	Random selection	Random without duplicates
INV	Invert a matrix	Least squares matrix solution
<		Less than
LEQ		Less than or equal to
=		Equal to
NEQ		Not equal to
GEQ		Greater than or equal to
>		Greater than
SET		Is a member of
AND		True if both true
OR		True if either true
NAND		False if both true
NOR		False if either true
NOT	True if false	
SIZE	Request dimension of	Create with dimension of
,	Turn into a vector	Join together
FLIP	Reverse order of elements	Rotate elements as specified
SWAP	Transpose an array	Selective transpose
TAKE		Select from start or end
DROP		Discard from start or end
/		Select if true, else compress
\		Select if true, else expand
SORT	Ascending sort	Asc. sort by collation sequence
TROS	Descending sort	Des. sort by collation sequence
ENC		Encode into a new number base
DEC		Decode from a new number base
FORM	Default numeric format	Specific numeric format
$	Default numeric i/o	Specific numeric i/o
IO	Default text i/o	Specific text i/o
EX	Execute text as numeric	Execute conditional on error

act as subroutines to others in the same workspace. The whole workspace can be saved somewhere when not in use or recovered with a single command. With our modern technology there is no limit to the physical size of this workspace and for most applications this makes files redundant.

FGL Variables

An FGL variable may be either text or numeric, and of any dimension. Zero-dimensional variables correspond to single characters or numbers. The numbers may be stored internally as binary, integer or floating point types, but that is transparent to the user. One-dimensional variables are lists or vectors; two-dimensional are tables or matrices; higher dimensions are allowed with no practical limit for more complex data storage needs. Data can be manipulated and inspected casually within the workspace. The underline symbol"_" causes the data on its right to be assigned to the location named on its left, while the use of the symbols "(;)" allows indexing. Arrays of more dimensions use more ";" symbols to separate the axes. A single quote is used to delimit entered text. If entered text should itself contain a quote, two adjacent quotes are used instead.

FGL Intrinsics

Most intrinsics can be applied directly to all or part of an array, in which case each term in the array is affected. Most intrinsics also have two uses: 'monadic', in which a variable is supplied as the right-hand argument, and 'dyadic', where a left and a right variable are supplied. Table 1 presents a fairly complete list of FGL intrinsics, divided into seven groups: arithmetic, algebraic, comparative, logical, manipulative, sorting/coding, and input/output. FGL intrinsics work on numeric and text arguments wherever possible, and as we said, you can throw multi-dimensional arrays at them and they'll respond appropriately.

FGL expressions may be compounded of intrinsics to any degree of complexity, the result from one operation being passed to the next as an argument. FGL lines are scanned from right to left. There is no precedence in order of execution between different intrinsics, even between multiply and add! The symbols "()" may be used to force execution to occur in any particular order. Unless the result of an operation is assigned to a name or used as the argument of another expression, it is displayed by default.

FGL intrinsics allow the user to do a great deal of serious work on data without ever writing a program. They've been carefully chosen so that you can do almost anything with them. Space doesn't permit a detailed discussion of each, to Table 2 presents some examples to give you the flavour.

FGL Operators

The syntax of the FGL intrinsics is that they can be 'monadic' or 'dyadic', taking data either as a right argument or on the right and on the left too. FGL operators are even more cunning. Instead of merely working on data as a right or left argument, they accept one or more intrinsics as right or left arguments, causing that intrinsic to be applied repeatedly to successive terms of an array. This sounds complicated, but actually the concept is very simple. There are five operators in FGL. They are called: reduction, scan, inner product, outer product, and axis.

There are about 40 intrinsics in FGL, and most can be applied either monadically or dyadically, giving about 70 possibilities. By using these intrinsics with operators, several hundred data manipulations can be specified without bothering to write a program. One of the most elegant aspects of FGL is the 'orthogonality' of its rules. In a sense an FGL operator is the high level language analogue of a machine code memory addressing mode, while an FGL intrinsic corresponds to a machine code instruction. Like a well-designed machine code instruction set,

Table 2 Examples of FGL Intrinsics

```
     + eric _ 10?100
34 12 57 89 39 4 67 33 79 44

     + eric _ eric - 19.5
14.5 ~7.5 37.5 69.5 19.5 ~15.5 47.5 13.5 59.5 24.5

     # eric
1 -1 1 1 1 -1 1 1 1 1

     MAX eric
15 ~7 38 70 20 ~15 48 14 60 25

     MIN eric
14 ~8 37 69 19 ~16 47 13 59 24

     ABS eric
14.5 7.5 37.5 69.5 19.5 15.5 47.5 13.5 59.5 24.5

     eric LIST 37.5 47.5
3 7

     bill _ 'the name of this language is FGL'

     bill LIST 'name'
5 6 7 3

     eric LEQ 20
1 1 0 0 1 1 0 1 0 0

     (eric LEQ 20) AND (eric GEQ 0)
1 0 0 0 1 0 0 1 0 0

     SIZE eric
10

     + tony _  2 6 SIZE 'hallo mother'
hallo
mother

     FLIP tony
 ollah
rehtom

     2 FLIP tony
llo ha
thermo
```

Table 2 (continued)

```
     SWAP tony
hm
ao
lt
lh
oe
 r

     0 2 DROP tony
llo
ther

     0 ¯2 DROP tony
hall
moth

     1 0 DROP tony
mother

     2 ¯3 TAKE tony
lo
her

     + paul _ LIST 10
1 2 3 4 5 6 7 8 9 10

     1 0 0 2 0 0 3 0 0 1 / paul
1 4 4 7 7 7 10

     1 0 0 1 5 1 0 0 0 1 3 1 0 0 0 1 1 1 \ paul
1 0 0 2 3 3 3 3 3 4 0 0 0 5 6 6 6 7 0 0 0 8 9 10

     + paul _ 10 ? 100
23 68 99 45 17 36 27 49 87 77

     SORT paul
5 1 7 6 4 8 2 10 9 3

     paul[SORT paul]
17 23 27 36 45 49 68 77 87 99

     paul[TROS paul]
99 87 77 68 49 45 36 27 23 17

     3 + EX '2+2'
7

     7 2 FORM 4.519 6.7
   4.52    6.70
```

FGL lets you employ (almost) any intrinsic with (almost) any operator on (almost) any array of data.

FGL Functions

An FGL function is composed of zero or more lines, each containing a valid FGL expression. Most programs in FGL are very short, up to about ten lines as a maximum. This contrasts strongly with the thousands of lines in programming languages such as BASIC (believed to stand for Boring And Senselessly Infantile Concept) or — and this one is very much for the history boys — some even more forgotten dialects such as COBOL (historians now think that this meant Central Official Bastion of Obsolete Languages). There are two reasons for this brevity. One is that the ability of the native FGL intrinsics and operators to work on whole arrays at a time means that loops, control structures and DO...WHILE instructions are hardly ever needed. The other is that instead of combining all a project's needs into one huge monolithic program, FGL breaks the job down into a set of sub-tasks, with a different and free-standing program responsible for each area.

FGL is rather like FORTH in this respect. You can define a vocabulary of your own FGL "verbs", "nouns" and "adjectives" to produce a personalised command syntax appropriate for any particular job. It takes about three minutes of programming in FGL to construct the functions which make the following line of English text a valid expression in FGL immediate execution mode:

Average Salary where (Sex is Female) and (Department is Accounts)

and which will produce the correct answer to this and any similar information request. The response will be almost instantaneous irrespective of the size of the personnel file, because this data consists of FGL vectors sitting in main memory all the time — there are no files involved.

FGL functions are in some respects similar to intrinsics. They may be monadic or dyadic, but they may also be 'niladic' — using no arguments at all. A niladic FGL program corresponds to the idea of a BASIC program. A monadic or dyadic program corresponds to the idea of a BASIC subroutine, but the FGL versions are free-standing and much more flexible. The following additional symbols may be used in FGL programs:

∧	To start or end function editing
"	The rest of the line is a comment
@	Branch to indicated line/label, or ignore if none specified
&	Statement separator

Lines may be prefaced with text labels terminated by a ":" symbol. Label names are automatically assigned their run-time line numbers as integer values, so these become pseudo-variables that may be referenced but not altered in the user's expressions.

Branching in FGL is of interest. The "@" branch expects to see an integer argument representing a valid line number. This integer may be calculated any way the user wishes. Multiple conditional branches similar to the CASE constructs of other languages become trivial in FGL. If the integer is not presented (by providing an "empty vector" of zero length) then FGL will ignore the branch instruction. The user may use any one of many FGL techniques to implement a conditional branch using this feature. Table 3 contains some examples of FGL functions.

FGL Commands

An FGL command generally acts immediately on the workspace or the long-term storage medium and affects the environment within which the FGL programmer is working. Aspects such as changing the number of columns in the display device line width, inquiring after the names of variables and functions etc. are typical uses of these system commands. Commands are prefaced

Table 3 Examples of FGL Functions

```
     ^ result _ average data
[1]  "This function will average a vector of numeric values
[2]  result _ (+/data) % SIZE data
[3]  ^

     average 1 2 3 4
2.5
     3#average 1 2 3 4
7.5

     ^ analysis
[1]  "This function will do some statistics on typed-in data
[2]  "Notice that it in turn uses the pre-written function 'AVERAGE'
[3]  'Please enter your data, separated by space(s), terminated by RETURN'
[4]  hisdata _ $
[5]  'Thank you'
[6]  'There were ';SIZE hisdata;' values entered'
[7]  'The largest was ';MAX/hisdata
[8]  'The smallest was ';MIN/hisdata
[9]  'The average was ';average hisdata
[10] 'Goodbye'
[11] ^

     analysis
Please enter your data, separated by space(s), terminated by RETURN
$:   50 30 40 20 60
Thank you
There were 5 values entered
The largest was 60
The smallest was 20
The average was 40
Goodbye

     ^ newtab _ crosstab oldtab
[1]  "This function cross-tabulates a table (row and column sums)
[2]  newtab _ oldtab,[1]+/[1]oldtab
[3]  newtab _ newtab,[2]+/[2]newtab
[4]  ^

     crosstab 4 4 SIZE LIST 16
 1   2   3   4  10
 5   6   7   8  26
 9  10  11  12  42
13  14  15  16  58
28  32  36  40 136

     ^ r _ factorial b
[1]  "Recursive function for calculating factorials
[2]  @(1 = r _ b)/0
[3]  r _ b # factorial b - 1
[4]  ^
```

with a leading")" symbol which acts as a delimiter to distinguish them from user objects which might have the same name. Commands are not generally included inside user's functions, but they may be so incorporated by enclosing them inside quotation marks and executed at run-time using the "EX" intrinsic.

FGL Extensions

The "$" symbol is used by FGL for two tasks: as primitive input-output intrinsic for numeric data, and also as a delimiting prefix to introduce an implementation-dependent additional FGL facility. The FGL syntax has been standardised by the ISO but vendors may offer any number of extra "goodies" in their own FGL interpreters. In order to minimise the problem of name clashes, these extensions are prefaced by "$", hence the name "dollar-functions" and "dollar-variables".

FGL Files

Because typical workspaces of 20 Giganodes (comparable in 1980's terms to a user having 200 Megabytes of physical RAM driven by about 10 dedicated 32-bit processors) are the norm these days, files play a rather minor role in FGL. Dedicated applications tend to store all the data in the workspace, while multi-user tasks employ the concept of the 'shared rodent interface' to squirt data between other active workspaces and user I/O devices such as mice, without bothering to use files. However, files can be used if required.

An FGL file is best thought of as a collection of adjacent elastic boxes. The boxes are numbered sequentially, starting at 1, with no maximum limit. A box can contain any FGL object of any size: a single number, a vector of 100,000 numbers, a text matrix of 10,000 rows by 100 columns, or a function. Depending on the implementation, dollar-functions may exist to allow a heterogeneous assortment of variables and functions to be grouped into a single package and filed as a single object. Filed objects may be replaced by larger or smaller ones without restriction. Acess is by box number: direct and sequential access of any type is therefore supported.

Whereas the conventional languages of the 1980's expected data records to be stored as 'horizontal' slices of a table, with one stored record per person or product in the data file, FGL records are normally stored 'vertically', with one boxed object per parameter in the data file. All the SALARIES go into box number 1; all the DEPARTMENTS into box 2; all the JOINING DATES into box 3, and so on. Files are therefore typically 'inverted' all the time, which usually obviates the need for a database.

Why FGL overtook BASIC

The overriding advantages of FGL were as follows:

- Extremely fast project development time
- A workspace in which to bring together programs and data
- A rich set of standard data handling primitives
- Ability to work on arrays without looping
- Modular function style with elegant control flow
- No limit to the range of applications that could be handled
- Sophisticated features like recursion for expert systems
- Standardised full-screen controls
- Standardised printer attribute handling
- Standardised colour graphics routines
- Shared rodent interface for control of mice
- Shared stereo interface for voice recognition and output
- Required no special hardware for its character set
- Upper & lower case characters for commercial applications
- Numeric precision to 15 significant figures

— A structure which made database handlers unnecessary
— Parallel processing as an integral part of the language

In addition, the existence of the earlier APL language meant that there was no need to create a new language interpreter for each implementation: simple modifications to the character input/output routines were all that was needed to convert APL into FGL.

Why FGL was well suited for fifth generation projects

The flirtation with knowledge-based expert systems which began at the start of the 1980's produced proposals for the deployment of other languages for expert systems applications, such as LOGO, LISP and PROLOG. These languages were strong on inference but weak on arithmetic. For example, PROLOG was good at solving the following types of problem:

'If time flies like an arrow, do fruit flies like a banana?"

In PROLOG this would be stated as:

&. Add (Time flies-like an-arrow)
&. Does (Fruit-flies like a-banana)

LISP was good at jobs like proposing robust hyphenation rules for word processing spelling checkers, rules which helped one to avoid infelicitous proposals for hyphen insertion in a word such as "toadstool". In LISP the code to do this would be stated in the user-friendly formulation:

((((((((((((((((toads(tool))))))))))))))))

LOGO excelled at helping people create random artwork designs using turtles; in fact it wiped out all competition in this field and ended up being referred to the Anti-Trust (Monopolies) Commission in the Galapagos Islands.

However, it eventually dawned on people that the basis for good expert systems design had to be a good general purpose computer language: one that was strong on logic and text manipulation while also being extremely fast at computation by the use of parallel processing. As parallel array processors emerged, moving away from the so-called "pipeline" processing concept to the aptly-named "dense-pack" synchronous computation field, the advantages of array-oriented languages for expert system design soon became irrestible. Declarative subsets of FGL soon emerged which modelled all the best features of the specialised expert systems languages without throwing away the indigenous ability to calculate (for example) antilogarithms.

The implications of Ashby's Law ("The variety of a controller must be as great as that of the system under control") meant that any attempt at artificial intelligence in the strict sense required the creation of an artificial model of reality that was robust for the uses to which it was being put. Digital implementations of this concept were inappropriate because of the sequential, clocked nature of their computations: even at sub-picosecond switching speeds, the variety was just not there. By the 1980s we had learnt that the answer lay in marrying microelectronic fabrication disciplines to the highly parallel, asynchronous nature of mental processes as implemented via neurons, axons and synapses in the human brain in an essentially analogue fashion. This was first done by a small company in Rockville in 1986, using protein instead of silicon as a semiconducting substrate. The rest, of course, is history.

However, the acceptance that true artificial intelligence required designers to step outside the sequential von Neumann architecture was enough to encourage the development of pseudo-intelligent systems. The goal of genuine understanding was set aside for the more prosaic (but commercially desirable) aim of flexibility and knowledge. FGL turned out be an excellent vehicle

for specifying heuristics drawn from fuzzy set theory, from catastrophe theory and from Bayesian statistics. These were robust means of integrating the large body of detailed knowledge that was available to the "domain specialists" who were implementing the systems. In particular the creation of off-shoots of FGL such as ESP (Expert System Producer), ESPRIT (Expert System Producer for Rapid Integration of Thoughts) and ESPION (Expert System Producer for Intelligence, Observation and Neutralisation) were formative in FGL's rapid penetration of this huge arena.

FGL was fast, flexible and mature. It was very good at Boolean logic. It handled recursion elegantly. It coped beautifully with problems requiring very rapid computation over vast memory areas. FGL programs could with trivial ease modify themselves or other FGL programs while executing. Most important of all, it was there and it worked. In retrospect, there was no other choice.

The reaction from the BASIC community

The response from existing BASIC users represented one of the decade's most fascinating examples of techno-cultural re-adjustment. Fortunately the accounts of this process were well documented at the time and we now know fairly precisely what happened. The first paper on FGL entitled simply "FGL: Fifth Generation Language" was published in a magazine of the time called 'BYTE' (now replaced by the hourly circulation electronic journal 'GNAW' — Gripping News Adventurously Written) in the 1983-84 period and the first implementations of the language became available early in 1984. To users of BASIC, who had been fiddling around with things like PASCAL, ADA, PROLOG, LISP, LOGO, FORTH, HOPE, SNOBOL, FORTRAN, RPG, COBOL, ALGOL, PL/1 etc. in the hope of finding something useful, the response was overwhelming.

Massed bands of cheerleaders ran through the streets of the world's major capitals holding placards with FGL programs written on them. Spectacular May Day parades in Red Square featured elaborate floats in the form of FGL 5-dimensional arrays. The Space Shuttle (not to be confused with the recently introduced no-reservation required Venus shuttle) was employed to blanket the ionosphere with FGL Reference Guides multiple-independently targeted on a re-entrant orbit to the world's computer conurbations. No resistance was encountered anywhere, except from those who had previously encountered APL. It was a landslide.

The reaction from the APL hostility

The first formal presentation of FGL to the APL hostility took place with the submission of the same paper to the annual APL conference held in Helsinki, Finland, June 11th — 15th 1984. Despite the better judgement of several of the paper's referees, the paper was accepted as part of the formal proceedings. This was a pity in many ways since previous conferences had tended to indicate that the pirate presentation of a rejected paper attracted a larger audience than one that had been accepted. Nevertheless, at least two people were observed to enter the room in which the paper was being delivered, one of whom was known not to be the paper's author. After the paper had been presented, the following questions were posed:

Q. "We already have proposals for an ASCII notation for APL. Why make such a big thing of it?"
A. "Because if APL is to take over in the disguise of FGL, we must kill APL notation completely for all except academic and research purposes. Nobody is prepared to rush out and buy new APL screens if they already own ASCII versions — and by now, they all do!"

Q. "Why not adopt one of the existing ASCII mnemonic standards?"
A. "You're missing the point! FGL will be used by people who have never seen APL notation and who never will! Who wants to talk about $RHO?"

Q. "You've destroyed the whole spirit of APL"
A. "FGL training books will contain references to APL notation in an appendix at the back. People who want to play with APL will still be able to. Most APL users care far less about the notation than most of the people who set APL standards. Try actually using FGL for a bit — you'll find that the APL spirit is still very much there."

Q. "What is the main objective behind such a major change?"
A. "First and foremost, to stop APL inexorably disappearing. This is happening partly because commercial pressures increasingly preclude the implementation of an overstruck character set, and partly because APL's funny symbols simply put most people off. As a percentage based on the language used per installed computer (including microcomputers), APL usage has been declining exponentially for 7 years and is now indistinguishable from the zero axis. If we don't adopt something like FGL now, APL will sink without trace."

Q. "Are there any other advantages?"
A. "Of course. FGL lets you use uppercase and lowercase in a standard fashion — vital for business applications. Suddenly, you can choose any screen or printer you want. It's now implemented on virtually any computer you want. And the hardware is ridiculously cheap, because everyone is using it. Also since you know APL already, you've already learnt FGL."

Q. "What about nested arrays?"
A. "This has to be the world's most irrelevant question! APL is already so much more powerful than BASIC that any further features would virtually constitute overkill. However, it has to be admitted that list structures come in handy for expert systems, so (provided no new symbols are needed) there's no reason why FGL cannot adopt whichever nested array syntax APL eventually decide on."

Q. "My IBM 3270-type terminal creates APL characters without overstrikes, so what's the problem?"
A. "IBM's EBCDIC code does include APL and non-APL characters in a single character set. However, IBM and plug-compatible manufacturers are the only companies that use EBCDIC; IBM has already started selling an ASCII-type screen, and it's now only a question of time before ASCII-based protocols start to muscle EBCDIC code away."

Q. "If the next generation of microcomputers are going to have memory-mapped screens, what's the problem?"
A. "You still won't get APL characters without a big struggle. Let's change now while there's still time."

Q. "What can I do in practice to help FGL emerge?"
A. "Write to the FGL Action Group, care of the author."

Table 4 depicts the relationship of the defunct APL symbols to the FGL versions that we all now use. The few APL fanatics who remained unconvinced were eventually discovered in their caves in the Adirondacks and the Hebrides. Lucrative offers were made to them and you can still visit them now; either as part of the 'living history' permanent exhibition in Williamsburg, Virginia, for the entertainment of heads of state, or (in the more traditional British style) in glass cases at the Science Museum in London. You can find them there located next door to James Watt's steam engine; if you press the button on the outside of the stand, they wake up and demonstrate

APL characters in semaphore — as a one-line function with no comments.

Footnote: This article was written by an FGL module entitled ESCRIT, Expert Systems Creation of Random Intelligent Text. For this reason any views expressed herein remain those of the system and do not necessarily represent opinions held by the author or any organisation with which it is associated.

Table 4 — APL/FGL Equivalents

The standard APL character set was evolved for paper printing terminals for which no penalty (apart from print speed) is attached to the concept of backspacing and overstriking a character to form a new composite symbol. The APL character set therefore contains more than the standard 96 ASCII printing characters. Despite this, APL does not contain the lower case alphabet, a serious deficiency. Furthermore, various punctuation symbols which also exist in ASCII are located in non-standard positions in the APL code table. The mapping of APL to FGL characters is as follows:

a. APL A-Z becomes FGL a-z
b. APL A-Z becomes FGL A-Z
c. APL 0-9 stays as FGL 0-9
d. APL punctuation is replaced by ASCII versions in FGL, if they exist
e. ASCII punctuation which is undefined in APL is redeployed in FGL
f. Reserved FGL upper case (APL underlined) names replace remaining symbols

This mapping can be installed on an APL interpreter without making any changes to the interpreter itself provided the following extensions exist:

— Ability to reconfigure input and output translate tables.
— Ability to create ambivalent user-defined functions.
— User-defined funtions allowed as arguments to operators.

This approach allows the current set of recommeded reserved words such as SIZE, LIST etc. to be replaced by national language variants if desired.

"..." indicates that the APL symbol is not supported in FGL. These cases do not affect any APL symbols in serious active use. The surprising aspect is how little change is actually required to APL; many of the APL symbols are a part of standard ASCII and hence FGL.

This first draft of the standard was part of the 1983 paper and some changes were of course made following comments on particular choices. Overall, however, the final standard bore a close resemblance.

APL	FGL	APL	FGL	APL	FGL
¨		?	?	⍀	\[1]
¯	...	⍵	...	⌿	/[1]
<	<	∈	SET	!	!
≤	LEQ	⍴	SIZE	⍪	,[1]
=	=	~	NOT	⍞	"
≥	GEQ	↑	TAKE	⍎	EX
>	>	↓	DROP	⍕	FORM
≠	NEQ	⍳	LIST	⍟	LOG
∨	OR	○	TRIG	⌹	INV
∧	AND	*	*	⍱	NAND
-	-	⍺	...	⍲	NOR
+	+	⌈	MAX	⍎⍕	...
×	#	⌊	MIN	⍋	LOCK
÷	%	_	...	⍒	IO
⍁	...	∇	^	⊖	FLIP
∆	@	Δ	...	⍉	TKOS
∘		∘	`	⍋	SORT
(	(	'	'	⊖	FLIP[1]
)	)	⎕	$	⍠	SWAP
[	[	⊂	...		
]	]	⊃	...		
/	/	∩	...		
\	\	∪	...		
;	;	⊥	DEC		
,	,	T	ENC		
:	:	|	|		
⊢	...	0-9	0-9		
⊣	...	A-Z	a-z		
◇	&	A-Z	A-Z		

TECHNICAL EDITORIAL

by Jonathan Barman and David Ziemann

Welcome to the technical section of VECTOR. In these pages we hope to present items of interest to technical APL readers. Therefore, in future issues of VECTOR you will find papers, articles, APL code, algorithms and more. There will also be a regular technical letters page and prize competition. We want to encourage you to contribute to any of the above and would particularly welcome letters and/or APL code, with a view to maintaining a dialogue on particular areas of interest.

In this issue we present four contributed papers. The first one is by Alan Hawkes, professor at the Department of Management Science and Statistics of the University College of Swansea, who shares with us some of his work in a paper entitles "Complex numbers in APL". As well as providing a listing of his complex number functions for APL without using a complex data type, he also gives a mathematical treatment of the algorithms that lead to them. Although written to run under MIPS APL, the code requires only minor alteration in one or two cases to be run on other implementations.

Next, David Doherty of the British United Provident Association (BUPA) has contributed "SCREENIO — a full screen manager". David charts the development of his screen design package, and shows how flexibility is preserved via the specification of a set of validation rules.

"Partitioning Data in APL" is submitted by Jonathan Barman. In it, Jonathan discusses a range of techniques for manipulating partitioned data within a standard VS APL environment. Useful partition functions are developed and explained in detail within the text.

Finally, David Ziemann presents his paper on the current draft of the international APL language standard. "Inside the international APL standard" takes an in-depth look at the way the standard works, what the actual content is, and how it might be useful to us.

PRIZE COMPETITION: This is Your Life

by David Ziemann

In the Game of Life a rectangular grid is used as a framework within which the behaviour of 'organisms' over successive generations is studied. Each square within the grid must be either filled or empty, representing either a live or dead/unborn cell. Rules that determine the conditions under which new cells are born and old ones die are used to produce the next generation of organisms. For example (and for interest only), the usual rules for birth and death are:

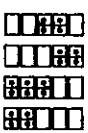
Birth — an empty cell becomes live in the next generation if it currently has exactly 3 live neighbours.

Death — a live cell dies in the next generation if:
- i) it currently has 0 or 1 live neighbours (isolation)
- or ii) it currently has 4 or more live neighbours (overcrowding)

In this context, each cell has 8 neighbours — the squares directly adjacent to it.

In APL, the most obvious data structure that can be used to represent the grid is a boolean matrix of known shape, with 1s indicating live cells and 0s dead ones. It is then possible to write an APL function that will take a boolean matrix right argument and return a new one that represents the next generation of cells. In fact this is quite an old problem, and it can be solved in a fairly satisfactory non-looping way. However, the time taken to calculate a successor generation is proportional to the grid size, and will be the same regardless of the contents of that grid. With large grid sizes, this cost can become prohibitive, and is particularly annoying when the grid only contains a smallish organism somewhere near the centre!

An alternative representation considers runs of 0s and 1s in the ravelled boolean matrix. For example, using this run-coding method, a 4 by 5 grid



would be represented by the integer vector: 2 2 4 5 2 2 3 where the quads represent empty cells and the dominoes live ones.

The first element of the run-code vector *always* indicates the leading number of 0s in the corresponding ravelled boolean matrix. The second element then gives the length of the next run of 1s, the third the length of the next run of 0s, and so on.

For typical Game of Life grids, this representation provides a reasonably compact data structure. Moreover, the time taken to calculate the successor grid is now proportional to the size of the contained organisms, rather than the grid size. This means that 'boring' generations are calculated quickly, whereas more complex ones take longer.

Because it is convenient to define and view a grid as a character matrix containing only spaces and crosses, the boolean representation is still important, and functions are required for converting between the boolean and run-coded forms. This is the subject of this issue's competition.

The task is to write the two monadic functions BTR and RTB. BTR converts its boolean vector argument into a run-coded vector, and RTB goes back the other way. Note that the shape of the grid is not relevant because BTR has a boolean vector argument and RTB a boolean vector result.

For example:

```
      +BM←4 5⍴0 0 1 1 0,0 0 0 1 1,1 1 1 0 0,1 1 0 0 0
 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

      ⍝ THE FOLLOWING IDENTITIES MUST ALWAYS HOLD TRUE:

      (⍴,BM)=+/BTR ,BM
1

      (,BM)∧.=RTB BTR ,BM
1
```

Each respondent must submit two APL functions that conform to the first ISO draft proposal for an APL language standard. (Don't worry — this effectively just means VS APL).

Additionally, respondents may include two further functions written for the APL implementations of their choice (please specify!). Note that no prizes will be awarded in this category.

Entries will be disqualified if they do not conform to the above descriptions, or if they rely on the external environment (ie, if a function is not a 'black box'). The criterion for judging functions that jump these hurdles will be the minimisation of the number of characters in the solution. This character count will only include the printing characters entered at the keyboard in del-editor mode, but will NOT include characters in the function header-line or in comment lines. Composite symbols count as one character.

A first prize of £30 (or equivalent) will be awarded to the author who, in the judges' opinions, submits the shortest correct entry. Two special commendation prizes of £10 each will additionally be awarded.

Many thanks to Paul Chapman, author of VIZ::APL, who suggested the competition topic.

COMPLEX NUMBERS IN APL

Alan G. Hawkes
Department of Management Science and Statistics.
University College of Swansea, U.K.

This paper describes a systematic method of representing complex numbers in standard APL in such a way that basic complex arithmetic and complex matrix algebra may be simply done. A listing of functions is given in section five.

The VECTOR typesetters were persuaded to resist the challenge of setting this article in print, and consequently it is reproduced directly from Alan's originals.

1. *Introduction: Representation of complex numbers*

One of the strengths of FORTRAN as a language for scientific and engineering computing is that it makes provision for declaring COMPLEX variables which may then be manipulated according to the rules of complex arithmetic by the standard algebraic symbols +, −, *, /. In addition, special functions or subroutines such as CEXP are available: in particular complex matrix algebra can be done by calling the appropriate subroutines.

Standard APL makes no provision for complex variables. After several proposals and discussions, Penfield (1979) produced proposals to extend the APL language into the complex field in a consistent manner. McDonnell (1981) describes such an extension which has been implemented in I.P. Sharp's APL. I would urge all writers of APL interpreters to follow suit as soon as possible; the scientists would then be more readily moved to change to APL.

In the meantime, it is quite easy to develop a simple means of handling complex numbers in standard APL. It could no doubt be done more elegantly in versions supporting nested arrays. I have not attempted to be exhaustive, but have restricted myself to the basis found necessary in my own applications. The functions mentioned in the text are listed in section five. They have been knocked together quite rapidly to solve a particular problem. The author would welcome amendments which would render them move elegant, efficient or accurate.

A single complex number $z = x + iy$ has real part x and imaginary part y (x and y both real), so that it may be thought of as an ordered pair of real numbers. We adopt the convention that a complex array of shape S will be represented by a real array of shape $2,S$ such that $Z \leftarrow X, [0.5]Y$ where X is the real part and Y the imaginary part (each of shape S).

Thus, for example

```
A←7 8 is a complex scalar 7+i8 with
REAL A
7
IMAG A
8
```

are real scalars corresponding to the real and imaginary parts of *A* (here *S* is the empty vector). In contrast

```
      □←B←2 1⍴7 8
7
8
```

is a complex vector with one element. Superficially

```
      REAL B
7
      IMAG B
8
```

appear to be the same as the previous case but this time 7 and 8 are one-element vectors (⍴*REAL B* and ⍴*IMAG B* are both 1).

More generally, the complex numbers 1+i5, 2+i6, 3+i7, 4+i8 may be assigned to an array of shape 2 4; so that

```
      C
1 2 3 4
5 6 7 8
```

is a four-element complex vector with

```
      REAL C
1 2 3 4
      IMAG C
5 6 7 8
```

An example of a complex 3 by 4 matrix, whose representation is an array of shape 2 3 4 is

```
      D
 1  2  3  4
 5  6  7  8
 9 10 11 12

13 14 15 16
17 18 19 20
21 22 23 24
```

with real and imaginary parts

```
      REAL D
 1  2  3  4
 5  6  7  8
 9 10 11 12

      IMAG D
13 14 15 16
17 18 19 20
21 22 23 24
```

Higher-dimensional arrays could be defined, but do not seem likely to be much used.

An alternative polar representation of a complex number

$$x + iy = re^{i\theta} = r\cos\theta + i\sin\theta$$

$$r = (x^2 + y^2)^{\frac{1}{2}}$$

is often useful. The modulus r and argument θ are obtained by functions *MOD Z* and *ARG Z*, while *R CART THETA* does the inverse transformation from polar to Cartesian co-ordinates. In this connection the functions *RAD* and *DEG* are useful to convert between angles measured in degrees and radians. *CONJ Z* yields the complete conjugate $X - iY$, while *COMPLEX X* converts a real array into a complex array with zero imaginary part, so that for example

```
      COMPLEX 3
3 0
```

is a complex scalar and

```
      COMPLEX 7 5
7 5
0 0
```

is a complex vector of length two.

2. *Complex arithmetic*

Two complex arrays of the same shape may be added or subtracted by $Z + W$ or $Z - W$ in the usual say. However, this does not satisfy the rule that if one of the arguments is a (complex) scalar it is repeated to become the same shape as the other argument and thus added to, or subtracted from, each element of the other argument. This property is a feature of the functions *CADD* and *CSUB* so, for example, using *C* from the previous section we have

```
      C CSUB 1 2
0 1 2 3
3 4 5 6
```

and we see that the complex scalar 1 + i2 is subtracted from each element of the complex vector *C*. The same is true for elementwise multiplication and division in the functions *Z CTIMES W* and *Z CDIV W* respectively. In each case *Z* and *W* must be the same shape or be complex scalars. As in the real case, (complex) vectors of length 1 and 1 by 1 matrices may act like scalars in this respect.

Simple monadic functions are *CRECIP Z*, the complex version of monadic ÷, *CEXP Z* and *CLOG Z*.

In principle it would be possible to write functions to manipulate complex arrays, such as *CTAKE* or *CDROP* etc., but one can always achieve the same ends by appropriate real manipulations.

3. *Complex matrix algebra*

The addition and subtraction of matrices whose elements are complex numbers is just a special case of the addition and subtraction of general arrays, as discussed in the previous section. Matrix multiplication is done by *Z CMM W*, which is the complex equivalent of *Z+.×W*, having equivalent rules about conformability and (complex) shape of the result.

This leaves three rather more difficult problems. The most important is that of finding the inverse of a complex matrix. This is an interesting problem which I discuss in this section. In the following section I deal with the

computation of the exponential of a matrix. The third problem is that of finding eigenvalues and eigenvectors of a complex matrix. I have not yet attempted this because it is the most difficult and because I have as yet not had any need to do so. However, it should be done and I would be pleased to hear from anyone who may already have done it.

The inverse of a complex matrix

In principle one could write a function which carries out some form of Gaussian elimination to find the inverse of a complex matrix. However, since APL is very good at inverting real matrices with ⌹ it seems a good idea to make use of that facility.

Denote the inverse of the complex square matrix $Z = X + iY$ by $U + iV$. Then, since their product must be the identity I, we have

$$XU - YV = I \tag{3.1}$$

$$XV + YU = 0 \tag{3.2}$$

Suppose there exist real λ_1, λ_2 such that

$B = \lambda_1 X - \lambda_2 Y$ is invertible. Let

$C = \lambda_2 X + \lambda_1 Y$.

Then $\lambda_1 \times (3.1) - \lambda_2 \times (3.2)$ and $\lambda_2 \times (3.1) + \lambda_1 \times (3.2)$ yield

$$BU - CV = \lambda_1 I$$

$$CU + BV = \lambda_2 I,$$

which can be solved to yield

$$U = (B + CB^{-1}C)^{-1} \ (\lambda_1 I + \lambda_2 CB^{-1}) \tag{3.3}$$

$$V = (B + CB^{-1}C)^{-1} \ (\lambda_2 I - \lambda_1 CB^{-1}) \tag{3.4}$$

If Z is non-singular there are at most $1\uparrow\rho X$ values of the ratio λ_1/λ_2 for which B is singular. Our general routine *CINV Z* therefore chooses λ_1, λ_2 at random. There will be a very small probability of hitting one of these 'eigenratios' . However, to be on the safe side, an error trap is set in line [7] which returns to choose another pair of values for λ_1, λ_2 if B should be singular.

Of course, if Z is singular, B will always be singular and we would get into an infinite loop. Thus we set a counter and switch off the error trap in line [10] after five attempts to find a non-singular B. Once a suitable B is found, the inverse $U + iV$ is constructed from (3.3) and (3.4) using only real matrix algebra. Error-trapping routines vary from one interpreter to another, so anyone using this function may need to make appropriate minor adjustments.

For particular problems one sometimes knows that the real part X of a matrix must be invertible. In that case one can take $\lambda_1 = 1$, $\lambda_2 = 0$ which imply that

$B = X$, $C = Y$ and thus

$$U = (X + YX^{-1}Y)^{-1} \tag{3.5}$$

$$V = -(X + YX^{-1}Y)^{-1}\ YX^{-1} \tag{3.6}$$

This solution is implemented in the function *CINVR Z*.

Similarly, if it is known that Y is non-singular, one may take $\lambda_1 = 0$, $\lambda_2 = -1$ so that $B = Y$, $C = -X$ which imply that

$$U = (Y + XY^{-1}X)^{-1}\ XY^{-1} \tag{3.7}$$

$$V = -(Y + XY^{-1}X)^{-1} \tag{3.8}$$

This solution is implemented in the function *CINVI Z*. A simple example in which neither the real nor imaginary parts are invertible is

$$\begin{pmatrix} 1 & 1 \\ 0 & i \end{pmatrix}^{-1} = \begin{pmatrix} 1 & i \\ 0 & -i \end{pmatrix} .$$

Any values of $\lambda_1, \lambda_2 \neq 0$ would suffice to invert this one. The general routine *CINV* does the job.

```
      CINV  2 2 2 ⍴ 1 1 0 0 0 0 0 1
1  0
0  0

0  1
0 ¯1
```

The matrix $\begin{pmatrix} 1+i & 2 \\ 1 & 1-i \end{pmatrix}$,however, is singular so

¯CINV 2 2 2 ⍴ 1 2 1 1 1 0 0 ¯1 yields the dreaded *DOMAIN ERROR*.

Note that any of these three inverse functions are generalisations of the real monadic ⌹ , except that we allow them to apply only to complex square matrices, vectors of length 1, or scalars. In the real case, of course, one gets a least squares fit for the non-square case. I have not attempted any generalisation of this feature.

4. *The exponential of a matrix*

In applied probability, and other areas of applied mathematics, one often needs to evaluate the exponential of a square matrix, defined by the infinite series

$$e^A = I + \frac{A^2}{2!} + \frac{A^3}{3!} + \ldots \quad .$$

The accurate and efficient computation of this is a rather delicate matter, see Moler and Van Loan (1978). I have adopted the simplest, and probably most effective, method of scaling and squaring. Consider the case of a real matrix A, which is implemented in the function *MEXP A*.

The most obvious thing to do is to truncate the series

$$e^A \simeq T_k(A) = \sum_{r=0}^{k} A^r \div r! \quad . \tag{4.1}$$

Bounds for the error of truncation are given by

$$|| \; T_k(A) - e^A \; || \leq \frac{||A||^{k+1}}{k+1!} \; [1 - ||A|| / (k+2)]^{-1} \quad , \tag{4.2}$$

where $||A||$ is the norm of the matrix. A number of different norms may be used. For simplicity of calculation use the 1-norm

$$||A|| = \max_i \sum_j |a_{ij}| \quad . \tag{4.3}$$

The scaling and squaring method, as implemented here, proceeds in three stages:-

(i) find the smallest non-negative integer M such that

$$||A \div 2^M|| = ||A|| \div 2^M < \tfrac{1}{2} ,$$

(ii) Approximate $e^{(A \div 2^M)}$ by the finite series (4.1), replacing A by $A \div 2^M$ and with $k = 13$ (truncation error less than 10^{-15}),

(iii) find $e^A = (e^A \div 2^M)^{2^M}$ by squaring M times.

For a complex matrix one simply needs to interpret $|a_{ij}|$ in (4.3) by the complex modulus, *MOD*, and replace +.× by *CMM* when performing matrix multiplication. This is done in the function *CMEXP*. For *MEXP* and *CMEXP* the inputs must be real, or complex as appropriate, square matrices, vectors of length 1 or scalars. In each case, the output has the same shape as the input.

Sometimes there are short cuts. For example, if I is a real identity matrix of the same shape as a real matrix Q and s a complex scalar, then

$$e^{-sI + Q} = e^{-s}\, e^{Q}$$

so that the exponential of this particular complex matrix can be written as the product of the exponential of a complex scalar and the exponential of a real matrix. In terms of the functions described in this paper this is equivalent to the rather less elegant form *CMEXP (COMPLEX Q) CSUB S CTIMES COMPLEX I* is equal to *(CEXP -S) CTIMES COMPLEX MEXP Q*.

Oh well, at least it's better than doing it in FORTRAN!

Finally, a word of warning. The obvious result

$$e^{A + B} = e^A\, e^B$$

is only true for matrices if A and B commute.

REFERENCES

FORBES, D. (1981), *Complex floor revisited*, APL81, APLQQ vol. 12 No. 1 September 1981, 107-111.

McDONNELL (1981), *An implementation of complex APL*, APLQQ vol. 11 No. 3, March, 1981, 19-22.

MOLER, C. and VAN LOAN, C. (1978), *Ninteen dubious ways to compute the exponential of a matrix*, SIAM Review, vol. 20 No. 4, 801-836.

PENFIELD, P. (1979), *Proposal for a complex APL*, APL79, APLQQ vol. 9 No. 4, June 1979, 47-53.

PENFIELD, P. (1981), *Principal values and branch cuts in complex APL*, APL81, APLQQ, vol. 12 No. 1, September 1981, 248-256.

```
      ∇R←ARG X;I
[1]   ⍝ ARGUMENT OF COMPLEX X, -PI<THETA≤PI
[2]    R←REAL X
[3]    I←IMAG X
[4]    R←(2×((I≥0)-0.5))×¯2○¯1⌈1⌊R÷((R*2)+(I*2))*0.5
       ∇

      ∇R←X CADD Y
[1]   ⍝ ADDITION OF COMPLEX X AND Y
[2]    R←((REAL X)+REAL Y),[0.5](IMAG X)+IMAG Y
       ∇

      ∇Z←R CART THETA
[1]   ⍝ POLAR TO CARTESIAN(COMLEX); THETA IN RADIANS
[2]    Z←(R×2○THETA),[0.5]R×1○THETA
       ∇

      ∇R←X CDIV Y
[1]   ⍝ DIVISION OF COMPLEX X BY Y
[2]    R←X CTIMES CRECIP Y
       ∇

      ∇R←X CTIMES Y;RX;IX;RY;IY
[1]   ⍝ MULTIPLIES ELEMENTWISE; X AND Y MUST BE SAME SHAPE OR COMP
      LEX SCALAR
[2]    RX←REAL X
[3]    IX←IMAG X
[4]    RY←REAL Y
[5]    IY←IMAG Y
[6]    R←((RX×RY)-(IX×IY)),[0.5](RX×IY)+(IX×RY)
       ∇

      ∇R←CRECIP X;RX;IX;MODSQ
[1]   ⍝ COMPLEX RECIPROCAL OF ARRAY X
[2]    RX←REAL X
[3]    IX←IMAG X
[4]    R←(RX÷MODSQ),[0.5]-IX÷MODSQ←(RX*2)+(IX*2)
       ∇

      ∇R←CEXP Z
[1]   ⍝ EXPONENTIAL OF COMPLEX Z
[2]    R←(*REAL Z)CART IMAG Z
       ∇
```

```
      ∇R←CINV Z;X;Y;B;C;I;U;LAMDA;D;S;J;S∆CINV
[1]   ⍝ INVERSE OF COMPLEX MATRIX WITH RANDOM TRANSFORM: AGH 1984
[2]    →(2≠1↑S←⍴Z)/MESS
[3]    →((1=⍴S)∨(2=⍴S)∧1=¯1↑S)/SCALAR
[4]    →((3=⍴S)∧0=-/1↑S)/CONTINUE
[5]   MESS:→0,⍴⎕←'ARGUMENT MUST BE COMPLEX SQUARE MATRIX'
[6]   SCALAR:Z←2 1 1⍴Z
[7]   CONTINUE:S∆CINV←1 2⍴COUNT,1
[8]    J←1
[9]   COUNT:→(5≥J←J+1)/RAND
[10]   S∆CINV←0 2⍴1
[11]  RAND:LAMDA←LAMDA÷+/LAMDA←?2⍴1000000000
[12]   B←(LAMDA[1]×X←REAL Z)-LAMDA[2]×Y←IMAG Z
[13]   C←(LAMDA[1]×Y)+LAMDA[2]×X
[14]   I←I∘.=I←⍳1↑⍴X
[15]   D←⌹B←(R←C+.×⌹B)+.×C
[16]   U←D+.×(LAMDA[1]×I)+LAMDA[2]×R
[17]   R←S⍴U,[0.5]D+.×(LAMDA[2]×I)-LAMDA[1]×R
       ∇

       ∇R←CINVI Z;Y;D;X;S
[1]   ⍝ INVERSE OF COMPLEX Z WHEN IMAG PART INVERTIBLE
[2]    →(2≠1↑S←⍴Z)/MESS
[3]    →((1=⍴S)∨(2=⍴S)∧1=¯1↑S)/SCALAR
[4]    →((3=⍴S)∧0=-/1↑S)/CONTINUE
[5]   MESS:→0,⎕←'ARGUMENT MUST BE COMPLEX SQUARE MATRIX'
[6]   SCALAR:Z←2 1 1⍴Z
[7]   CONTINUE:D←⌹Y←(R←X+.×⌹Y←IMAG Z)+.×X←REAL Z
[8]    R←S⍴(D+.×R),[0.5]-D
       ∇

       ∇R←CINVR Z;Y;D;X;S
[1]   ⍝ INVERSE OF COMPLEX Z WHEN REAL PART INVERTIBLE
[2]    →(2≠1↑S←⍴Z)/MESS
[3]    →((1=⍴S)∨(2=⍴S)∧1=¯1↑S)/SCALAR
[4]    →((3=⍴S)∧0=-/1↑S)/CONTINUE
[5]   MESS:→0,⎕←'ARGUMENT MUST BE COMPLEX SQUARE MATRIX'
[6]   SCALAR:Z←2 1 1⍴Z
[7]   CONTINUE:D←⌹X←(R←Y+.×⌹X←REAL Z)+.×Y←IMAG Z
[8]    R←S⍴D,[0.5]D+.×-R
       ∇


       ∇R←CLOG Z
[1]   ⍝ LOGARITHM OF COMPLEX Z
[2]    R←(⍟MOD Z),[0.5]ARG Z
       ∇
```

```
      ∇R←CMEXP A;J;T;M;S
[1]   ⍝ EXPONENTIAL OF COMPLEX MATRIX: AGH84
[2]    →(2≠1↑S←⍴A)/MESS
[3]    →((1=⍴S)∨(2=⍴S)∧1=¯1↑S)/SCALAR
[4]    →((3=⍴⍴A)∧0=-/1↑⍴A)/CONTINUE
[5]   MESS:→0,⍴⎕←'ARGUMENT MUST BE COMPLEX SQUARE MATRIX'
[6]   SCALAR:A←2 1 1⍴A
[7]   CONTINUE:A←A÷2*M←0⌈⌈1+2⍟1E¯10+⌈/+/MOD A
[8]    R←T←COMPLEX T∘.=T←⍳(J←1)↑1+⍴A
[9]   SERIES:R←R+T←T CMM A÷J
[10]   →(13≥J←J+1)/SERIES
[11]   J←0
[12]  SQUARE:→(M<J←J+1)/EXIT
[13]   R←R CMM R
[14]   →SQUARE
[15]  EXIT:R←S⍴R
      ∇

      ∇R←COMPLEX X
[1]   ⍝ CONVERTS REAL X TO COMPLEX
[2]    R←X,[0.5](⍴X)⍴0
      ∇

      ∇R←X CMM Y;RX;RY;IX;IY
[1]   ⍝ COMPLEX MATRIX MULTIPLICATION
[2]    RX←REAL X
[3]    IX←IMAG X
[4]    RY←REAL Y
[5]    IY←IMAG Y
[6]    R←((RX+.×RY)-(IX+.×IY)),[0.5](RX+.×IY)+(IX+.×RY)
      ∇

      ∇R←CONJ X
[1]   ⍝ CONJUGATE OF COMPLEX X
[2]    R←(REAL X),[0.5]-IMAG X
      ∇

      ∇R←X CSUB Y
[1]   ⍝ SUBTRACTION OF COMPLEX Y FROM X
[2]    R←((REAL X)-REAL Y),[0.5](IMAG X)-IMAG Y
      ∇


      ∇R←DEG X
[1]   ⍝ CONVERTS X IN RADIANS TO DEGREES
[2]    R←X×180÷○1
      ∇

      ∇R←IMAG X;S
[1]   ⍝ EXTRACTS IMAG PART OF COMPLEX X
[2]    →(2=1↑⍴X)/CONTINUE
[3]    →0,⍴⎕←'COMPLEX X MUST HAVE 2=1↑⍴X'
[4]   CONTINUE:R←S⍴(¯1,S←1↓⍴X)↑X
      ∇
```

```
      ∇R←MOD X
[1]   ⍝ MODULUS OF COMPLEX X
[2]    R←(((REAL X)*2)+((IMAG X)*2))*0.5
      ∇

      ∇R←MEXP A;J;T;M;S
[1]   ⍝ EXPONENTIAL OF REAL MATRIX: AGH84
[2]    S←⍴A
[3]    →(1=⍴,A)/SCALAR
[4]    →((2=⍴⍴A)∧(0=-/⍴A))/CONTINUE
[5]    →0,⍴⎕←'ARGUMENT MUST BE REAL SQUARE MATRIX'
[6]   SCALAR:A←,A
[7]   CONTINUE:A←A÷2*M←0⌈⌈1+2⊛1E¯10+⌈/+/|A
[8]    R←T←T∘.=T←⍳(J←1)↑⍴A
[9]   SERIES:R←R+T←T+.×A÷J
[10]   →(13≥J←J+1)/SERIES
[11]   J←0
[12]  SQUARE:→(M<J←J+1)/EXIT
[13]   R←R+.×R
[14]   →SQUARE
[15]  EXIT:R←S⍴R
      ∇

      ∇R←REAL X;S
[1]   ⍝ EXTRACTS REAL PART OF COMPLEX X
[2]    →(2=1↑⍴X)/CONTINUE
[3]    →0,⍴⎕←'COMPLEX X MUST HAVE 2=1↑⍴X'
[4]   CONTINUE:R←S⍴(1,S←1↑⍴X)↑X
      ∇

      ∇R←RAD X
[1]   ⍝ CONVERTS X IN DEGREES TO RADIANS
[2]    R←○X÷180
      ∇
```

SCREENIO — A FULL SCREEN MANAGER

by David Doherty

For some time now, BUPA has been a major user of APL. The majority of the APL applications which we write for our internal users are highly interactive in nature and to ensure that this interaction is as 'painless' as possible, we take full advantage of the IBM fullscreen facilities available to us under AP124.

Until SCREENIO, the implementation of these fullscreen systems relied solely upon a set of screen design and low-level screen handling software which had been developed within BUPA. This software is similar in nature to the IBM supplied screen software.

SCREENIO treats screens as self-contained entities and addresses such problems as dynamic screen output, complex validity checks, PF key handling and scrolling.

The simple parameter definitions which describe a SCREENIO screen, ensure that they are simple to build, maintain and modify.

Before describing SCREENIO, let us first consider some of the problems related to screens and their management. We shall also review the screen software we already had and examine how far it went towards solving those problems.

SCREEN MANAGEMENT

Designing a screen to run within any application system requires careful consideration and planning. It must be considered both from the aspect of the user and the application itself.

The Screen-User Interface

— Physical layout:
The physical layout of fields on the screen, ie, their position, shape, and logical order (I have not seen many screens where the 'first' prompt appears at the bottom left hand corner of the screen!). Physical layout is a vital aspect of screen design and can mean the difference between a screen being easy or difficult to use.

— Field intensity:
Skilful use of intensity can improve readability of the screen and highlight important screen details.

— Static text:
Fixed prompts and headings which guide the user in both the use and purpose of the screen, are vital and must be clear and concise.

— Dynamic text:
Anything from time and date displays to dynamic headings or error messages constitute dynamic text; these guide the user in his treatment of the screen.

— PF Keys:
They are useful for all manner of things eg, summoning help displays which describe the screen, signalling QUIT or SAVE, controlling vertical and horizontal scrolling, invoking special facilities peculiar to that screen or application.

The Screen-Application Interface

— Referencing screen fields:
Screen fields must be referenced in order to read or write to them.

— Validation:
Once the screen has been displayed, the user's entries are examined and validated.

— Screen handling:
After, for example, an invalid input, it is usual to re-position the cursor, display error flags or messages, sound the terminal alarm and so forth.

— Capturing screen input:
Finally, it is necessary to capture the users' (validated) inputs to the screen.

SCREEN MANAGEMENT — WHAT HAD BEEN DEVELOPED

Screen design software and low-level screen utilities were already in use...

Screen-User Interface

— Physical layout:
Our screen design software allowed easy definition and modification of the positions, shape and type of all screen fields.

— Field intensity:
Field intensity could be set dynamically as well as at the design stage.

— Static text:
Defined during the design stage.

— Dynamic text:
Low-level software allowed us, under the control of application code, to write to the screen.

— PF keys:
The key depressed may be detected, but handling of the key has to be controlled by application code.

The Screen-Application Interface

— Referencing screen fields:
Field labels, which simplify field referencing, are defined during the design stage.

— Validation:
All validation must be handled by application code.

— Screen handling:
Screen control is effected through low-level software driven by application code.

— Capturing screen inputs:
The capturing of screen inputs must be controlled by application code.

Clearly, the management of the screen requires a combination of calls to the low-level screen handling functions for comunication with and manipulation of the screen, coupled with application code to actually govern the handling of the screen.

SCREEN MANAGEMENT — A CHANGE IN APPROACH

From careful consideration of the above, the advantages in treating screens as self-contained entities which could be handled by standard software were recognised, thereby:

- Removing the repetitive aspects of the process, saving development time and eliminating redundant code.
- Combining both the definition of the physical attributes of the screen with dynamic output, verification, error messages and PF Key handling, so that the screen could be regarded as a single entity.
- Providing a standard way of defining these 'facets' of the screen so as to simplify its maintenance.

It would then be possible to drive screens via a standard screen manager which would only require the name of the screen to be used. The manager would then drive the screen, performing all validation, executing any dynamic outputs and handling the PF Keys according to the parameter specifications made at the design stage.

One possible problem with this approach might be that the programmer would be constrained by such a package. Flexibility was therefore given a high priority.

SCREENIO — A POSSIBLE SOLUTION

With these aims in mind, SCREENIO evolved. Logically, it is divided into two groups — maintenance and application. The maintenance routines (which are themselves SCREENIO screens!) are held on a central library and loaded as necessary.

Within the application system only the application group is needed to drive the screens and should be stored as a part of the application.

Maintaining the Screens

SCREENIO screens are held on file and are referenced by name. Its screen maintenance routines utilise our existing screen design software: they do not supplant it.

The creation or amendment of a SCREENIO screen involves extra steps in the design stage. These extra steps are integrated with the existing screen design software.

SCREENIO's parameter definitions consider the screen both from the aspect of the user and the application.

The Screen-User Interface

- Physical layout: No change.

— Field intensity: No change.

— Static text: No change.

— Dynamic text:
Text may be written to any field on entry to the screen. An OUTPUT PARAMETER is simply defined for that field. Thus dynamic headings, time, help and date displays are catered for.

— PF Keys:
They may be separately defined for each screen. Some PF keys are defined according to our own internal standards. These are:

PF1 summons a help display.
PF3 is QUIT.
PF12 is END.
PF9 displays the PF key descriptions
PF keys 7, 8, 10 and 11 are reserved for vertical and horizontal scrolling of data.

These keys, along with their standard descriptions, are automatically defined for each screen when it is created. Scrolling will only operate if the data exceeds the size of the field in either, or both of, the vertical and horizontal dimensions.

The remaining keys, 2, 4, 5 and 6 may be defined by the programmer as special purpose keys. Both a description and a definition may be entered. EXECUTABLE EXPRESSIONS are entered for the definitions and text for their descriptions.

The Screen — Application Interface

— Reference screen fields: No change.

— Validation:

SCREENIO provides for the verification of screen input in two ways:

INDIVIDUAL FIELD VALIDATION

Fields may be validated on an individual basis. For each input field, a number of parameters are defined during the design stage and they control the validation for that field.

For each field, an (EXECUTABLE) VALIDATION EXPRESSION is entered, which is used to validate the contents of the field. SCREENIO contains some 'in-built' validation routines for the more common validation requirements, like numeric validation, table lookups etc. They are invoked using special characters within the EXPRESSION.

If the input into the field is INVALID, SCREENIO will write an ERROR MESSAGE which the programmer has defined, to a nominated ERROR field. The ERROR MESSAGE may be static, or dynamically defined.

The input fields can be flagged as MANDATORY or OPTIONAL which means that the programmer can force the user to input details where necessary or omit them when defaults have been predefined.

Similarly, SCREENIO can be directed to CLEAR INVALID entries from input fields or leave the entry for the user to edit.

Finally, SCREENIO will assign the contents of the field to a (GLOBAL) variable which the programmer nominates.

RELATED FIELDS

It is frequently necessary to consider fields in relation to one another; for example, it may be necessary to multiply two fields together to form a new result, which may in turn have to be validated.

SCREENIO provides for this through the entry of 'rules' which apply to the input fields. Each 'rule' may apply to as many, or as few fields as required. Like the individual field validation, the programmer defines several parameters.

An EXECUTABLE VALIDATION EXPRESSION is used to examine input fields in relation to one another.

If the result is VALID, an (optional) result may be written to a designated field. If the result is INVALID, an ERROR MESSAGE which again may be static or dynamic is written to an ERROR FIELD.

— Screen handling:
SCREENIO handles the screen in a consistent and logical manner following the use of PF keys, error messages etc.

— Capturing screen inputs:
SCREENIO assigns (as described above) all the users' inputs into (GLOBAL) variables.

Driving the Screen

To run a SCREENIO screen it is only necessary to call SCREENIO with the name of the required screen passed to it as an argument.

It utilises the low level screen handling software, governing it according to the operation being performed. We may summarise the execution of SCREENIO as follows.

1. Reads the file-held variables for the required screen into the workspace.
2. Formats the screen and writes both the STATIC and DYNAMIC text.
3. Displays the screen and awaits user input.
4. Actions key depressed.
5. Validates input and assigns it to (GLOBAL) variables.
6. The screen is finally left. The key depressed is returned as an explicit result.

The application code can act upon the key returned and utilise the GLOBAL variables SCREENIO has defined.

CONCLUSIONS

The effort put into SCREENIO has proved worthwhile. It has shown itself to be a valuable tool which meets its design criteria.

By treating screens as single entities it provides us with a coherent approach to the use of screens within applications and the problems encountered in their design use.

It is easy to use, requiring only a single call to SCREENIO within application code.

It is eminently flexible, controlling dynamic output to the screen, PF key handling, validation of fields both individually and in relation to each other, and finally, returning all input from the screen in a convenient collection of variables.

Above all, SCREENIO does not constrain the programmer because it allows EXECUTABLE EXPRESSIONS within the parameter definitions.

We are not, however, complacent. SCREENIO evolved into its present state and is continuing to evolve as the problems associated with screens and their management are reconsidered, new problems encountered or new approaches sought.

APL AND PARTITIONED DATA

by Jonathan Barman

Introduction

APL arrays provide a natural way of partitioning data. A matrix can be viewed as a set of vectors; each row of a matrix of numbers could be vectors of costs incurred by each department in a business. Adding up the total costs incurred by each department is then a simple matter of applying plus reduction along the last dimension. Reduction and Scan operators allow the application of any scalar function along any axis of an array, and provide powerful tools for creating functions which work on partitioned data. There are, however, limitations in some applications. In the example of department costs, some departments may incur many cost items, and have long vectors, while others may have only one or two cost items. APL arrays have to be rectangular, so holding the costs as a matrix means that all the rows have to have the same length; short vectors have to be padded out with zeros to match the largest number of cost items. The amount of padding required can be so large that it becomes difficult to manipulate the matrix without workspace full messages, although the amount of data is relatively small. If one department out of 100 departments had 5000 cost items and all the other departments averaged 5 items apiece, then the matrix has to be 100 rows by 5000 columns taking up 2,000,000 bytes, of which only 40,000 bytes is data.

Another difficulty is where the data is normally manipulated as a vector and it is inconvenient to form it into a matrix so that reductions and scans can be applied, and then reformat it as a vector. For example, text typed by the user of a system may need to be manipulated and re-displayed, and it is convenient to keep the text as a vector throughout the processing.

This article explores the ways in which partitioned data can be processed in a more natural way, without looping. The techniques are well known and have been in use for many years. The Working Memorandum on Boolean Techniques by Robert A. Smith was published by STSC in 1975, and sets out the fundamental ideas and lists an extensive set of functions. The FinnAPL Idiom list contains examples of manipulating partitioned data. The APL*Plus and Sharp timesharing services both provide workspaces of partition functions.

Before plunging into detail, the general principles will be illustrated with a simple example. The principles will then be analysed in more detail and illustrated with more examples.

Taking the department cost example, assume that each department has a unique code and that the costs and codes are held in two numeric vectors COSTS and DEPTS. The costs and department codes were entered from invoices, so that each cost has a corresponding department code.

There are three basic ways of adding up the costs for each department; by looping through each department code, by forming the data into a matrix and using plus reduction, or by using partition techniques. The looping method could be implemented as follows:

```
      ∇R←DEPTS ADDUP COSTS;A;B;⎕IO
[1]   ⍝ ADD UP <COSTS> FOR EACH CODE IN <DEPTS>
[2]   ⍝ <R[;1]> IS DEPT CODES, <R[;2] IS TOTAL COSTS.
[3]   ⍝ LOOPING METHOD.
[4]    ⎕IO←1
[5]    R←0 2⍴0
[6]   L1:→(0∈⍴DEPTS)/0
[7]    A←1↑DEPTS
[8]    B←A=DEPTS
[9]    R←R,[1]A,+/B/COSTS
[10]   B←~B
[11]   DEPTS←B/DEPTS
[12]   COSTS←B/COSTS
[13]   →L1
       ∇

       1 4 4 3 1 ADDUP 10 20 30 40 50
  1 60
  4 50
  3 40
```

This method is inefficient if large amounts of data are involved. Lines 9 to 12 reassign the variables, so data is being moved in memory for every unique department found.

Forming the data into a matrix is more efficient than the looping method, but, as explained above, there may be workspace full problems:

```
      ∇R←DEPTS ADDUP COSTS;A;P;⎕IO
[1]   ⍝ ADD UP <COSTS> FOR EACH CODE IN <DEPTS>
[2]   ⍝ <R[;1]> IS DEPT CODES, <R[;2] IS TOTAL COSTS.
[3]   ⍝ MATRIX METHOD.
[4]    ⎕IO←1
[5]    ⍝ SORT INTO DEPT CODE SEQUENCE
[6]    A←⍋DEPTS
[7]    DEPTS←DEPTS[A]
[8]    COSTS←COSTS[A]
[9]    ⍝ FIND WHERE CODES CHANGE
[10]   P←DEPTS≠1↓DEPTS,0
[11]   P[(0≠⍴P)/⍴P]←1
[12]   ⍝ FIND NUMBER OF CODES FOR EACH DEPT.
[13]   A←P/⍳⍴P
[14]   A←A-¯1↓0,A
[15]   ⍝ FORM EXPANSION VECTOR.
[16]   A←A∘.≥⍳⌈/0,A
[17]   ⍝ MAKE COSTS INTO A MATRIX.
[18]   R←(⍴A)⍴(,A)\COSTS
[19]   ⍝ ADD UP, AND APPEND DEPT CODES.
[20]   R←(P/DEPTS),[1.5]+/R
       ∇

       1 4 4 3 1 ADDUP 10 20 30 40 50
1 60
3 40
4 50
```

Forming the data into a matrix requires a technique which is constantly being used when dealing with partitioned data. Line 10 is a "not-equals positive difference operation", and line 14 is a "minus negative difference operation".

Lines 10 and 11 generate a "partition vector". Line 11 is necessary because it cannot be guaranteed that a department code of zero does not exist. If the rotation method is used:

```
P←DEPTS≠1⌽DEPTS
```

then line 11 is required in case there is only one department code in the data.

Care has been taken that empty arguments do not cause an error. Line 11 checks for an empty vector. Line 14 could have been written as:

```
A←A-0,¯1↑A
```

which would have caused a length error if A was empty. Line 16 has a 0 catenated to A in case it is empty. It is good practice to ensure that all code will work properly on empty vectors, but it is sometimes simpler to branch out on empty at the beginning of the function rather than having to include special processing as in line 11.

The partitioned data approach is as follows:

```
      ∇R←DEPTS ADDUP COSTS;A;P;⎕IO
[1]   ⍝ ADD UP <COSTS> FOR EACH CODE IN <DEPTS>
[2]   ⍝ <R[;1]> IS DEPT CODES, <R[;2] IS TOTAL COSTS.
[3]   ⍝ PARTITION METHOD.
[4]    ⎕IO←1
[5]   ⍝ SORT INTO DEPT CODE SEQUENCE
[6]    A←⍋DEPTS
[7]    DEPTS←DEPTS[A]
[8]    COSTS←COSTS[A]
[9]   ⍝ FIND WHERE CODES CHANGE
[10]   P←DEPTS≠1↓DEPTS,0
[11]   P[(0≠⍴P)/⍴P]←1
[12]  ⍝ CUMULATIVE SUM FOR EACH DEPT.
[13]   R←P/+\COSTS
[14]  ⍝ CONVERT TO INDIVIDUAL SUMS.
[15]   R←R-¯1↓0,R
[16]  ⍝ APPEND DEPT CODES
[17]   R←(P/DEPTS),[1.5]R
       ∇

       1 4 4 3 1 ADDUP 10 20 30 40 50
1 60
3 40
4 50
```

The steps down to line 11 are identical to the matrix method. Line 13 gets the overall cumulative sum for each department, and line 15 does the "minus negative difference

operation" which converts the cumulative sums back to individual sums. This relationship between scan and negative difference operation is another important technique which will be explored more fully later.

The ADDUP function is really carrying out three processes: sort the data, set up a partition vector, and carry out a partitioned plus reduction. The processes are needed in many varied circumstances, so it is convenient, and better programming practice, to have separate functions. Lines 10 and 11 generated a trailing partition vector as it was needed in this form on line 13. A trailing partition vector is one where a 1 flags the end of each partition:

```
      A←2 2 2 6 6 7 7 7
      A≠1↓A,0
0 0 1 0 1 0 0 1
```

A leading partition vector is one where a 1 flags the start of each partition:

```
      A≠¯1↓0,A
1 0 0 1 0 1 0 0
```

All partition functions need a partition vector as an argument, and it is necessary to standardise on either leading or trailing partitions. As the literature on partition functions always uses leading partitions, we will do likewise. The first function to be defined is one to create a partition vector:

```
      ∇R←CREATE∆PARTITION A
[1]   ⍝ <R> IS A LEADING PARTITION VECTOR WITH 1'S WHERE'
[2]   ⍝ <A> CHANGES.
[3]    R←A≠¯1⌽A
[4]    →(0∈⍴R)/0
[5]    R[⎕IO]←1
      ∇

      CREATE∆PARTITION 1 1 1 1 8 8 50 50 50
1 0 0 0 1 0 1 0 0
```

Line 3 uses the rotation method to allow for the data being either character or numeric, line 4 branches on empty, and line 5 guarantees the first element is a 1.

```
      ∇R←P P∆PLUS∆RED A
[1]   ⍝ <P> IS A LEADING PARTITION VECTOR,
[2]   ⍝ <A> IS A NUMERIC ARRAY.
[3]   ⍝ <R> IS A PARTITIONED PLUS REDUCTION ON THE
[4]   ⍝ FIRST DIMENSION OF <A>.
[5]    R←(1⌽P)⌿+\A
[6]    R←R-(⍴R)⍴0,[⎕IO]R
      ∇
```

Lines 5 and 6 of the partitioned plus reduction function are generalisations of lines 13 and 15 of the last ADDUP example. The rotate of the leading partitions on line 5 changes them into trailing partitions, and the plus scan is carried out along the first dimension so that the data can

be a matrix. Line 6 carries out the "minus negative difference operation" along the first dimension of the array.

```
      P
1 0 0 0 1 0 1 0 0
      A
  6  3  1
  1  6  7
  1  4  1
  5  7  6
 10  9  6
  1  7  5
  8 10  8
  3  1  8
  4  7  8
      P P⍙PLUS⍙RED A
 13 26 15
 11 16 11
 15 18 24
```

A more general accumulation function can be written in place of the ADDUP function:

```
      ∇R←ACCUMULATE A;P;⎕IO
[1]   ⍝ <A[;1]> IS A SET OF CODES.  REMAINING COLUMNS
[2]   ⍝ OF <A> IS DATA.  <R> IS THE UNIQUE SET OF
[3]   ⍝ CODES IN COUMN 1, WITH THE TOTALS OF THE DATA
[4]   ⍝ IN THE REMAINING COLUMNS.
[5]    ⎕IO←1
[6]    →(0∈⍴R←A)/0
[7]    R←R[⍋R[;1];]
[8]    P←CREATE⍙PARTITION R[;1]
[9]    R←(P/R[;1]),P P⍙PLUS⍙RED 0 1↓R
       ∇

    A
5 37 25 99
4 76 66  8
4 89 28 44
4 48 24 28
2 17 49 90
5  7 91 51
   ACCUMULATE A
2  17  49  90
4 213 118  80
5  44 116 150
```

Difference Operations

A list of the boolean difference operations are given in the appendix. At first sight they tend to look similar, and it is difficult to appreciate which ones are going to be useful. Rather than go through them all, the following functions show how the more popular difference operations are used in practice.

The greater-than negative difference operation keeps the first one in each series of ones, and sets the remaining elements to zero:

```
      A←0 1 1 0 0 1 1 1 1 0
      A>¯1↓0,A
0 1 0 0 0 1 0 0 0 0
```

This difference operation is very useful when analysing text typed by the user. For example, when checking fullscreen data input it is usually necessary to count how many words or numbers have been entered in each screen field:

```
      ∇R←WORD∆COUNT A
[1]   ⍝ <A> IS A CHARACTER MATRIX.   <R> IS THE NUMBER
[2]   ⍝ OF WORDS OR NUMBERS ON EACH ROW OF <A>
[3]    R←A≠' '
[4]    R←R>(⍴R)↑0,R
[5]    R←+/R
       ∇

       A
 ONE   TWO
12 34 6
     589
      WORD∆COUNT A
2 3 1
```

When errors are found in the text typed by the user, an error message has to be displayed describing what has gone wrong. It is nice to be able to point out the exact location of the trouble:

```
      ∇R←ERR REPORT∆ERROR TEXT;A;CR
[1]   ⍝ <ERR> IS A BOOLEAN ERROR INDICATOR WITH AN ELEMENT
[2]   ⍝ FOR EACH WORD IN THE CHARACTER VECTOR <TEXT>.
[3]   ⍝ <R> IS AN ERROR MESSAGE.
[4]    A←TEXT≠' '
[5]    A←A>¯1↓0,A
[6]   ⍝ VSAPL CARRIAGE RETURN CHARACTER.
[7]    CR←0 1 0/⎕TC
[8]    R←'INVALID ITEM:',CR,TEXT,CR,A\ERR\'^'
       ∇

      0 0 1 0 REPORT∆ERROR ' ONE    TWO THRE   FOUR'
INVALID ITEM:
  ONE    TWO THRE   FOUR
         ^
```

Lines 4 and 5 flag the first letter of each word in the TEXT, which is used to position the caret on line 8.

Two algorithms for checking numbers were published in Quote Quad, and they both exhibit the use of difference operations. Algorithm 139 by Gerald Bamberger in the March 1980 issue of Quote Quad (Vol 10 No 3) verifies numeric input, and is similar to the Quad function available on Sharp APL and APL*Plus APL.

```
      ∇R←VI A
[1]   ⍝ VERIFY NUMERIC INPUT.
[2]   ⍝ <A> IS A CHARACTER VECTOR CONTAINING GROUPS OF
[3]   ⍝ CHARACTERS DELIMITED BY ONE OR MORE SPACES.
[4]   ⍝ <R> IS A BOOLEAN VECTOR WITH A 1 WHERE THE
[5]   ⍝ CHARACTER GROUP IS A VALID NUMBER.  NUMBERS
[6]   ⍝ OUTSIDE RANGE  (⌈/⍳0)<A<⌊/⍳0  COUNTED AS VALID.
[7]    R←' 1111111111234'[' 0123456789.¯E'⍳'0 ',A]
[8]    R←1+≠((R∈'234')∨R≠¯1+' ',R)/R
[9]    R←R∈(8 3⍴0 41 431)+1 12 121 21 31 312 3121 321∘.×1 100
       1000
       ∇

      VI'123 1.3  3¯4  3.4 3E4  ¯3E¯34  E3'
1 1 0 1 1 1 0
```

A slightly simpler version, excluding E notation values, may be preferred:

```
      ∇R←ΔVI A
[1]   ⍝ VERIFY NUMERIC INPUT.
[2]   ⍝ <A> IS A CHARACTER VECTOR CONTAINING GROUPS OF
[3]   ⍝ CHARACTERS DELIMITED BY ONE OR MORE SPACES.
[4]   ⍝ <R> IS A BOOLEAN VECTOR WITH A 1 WHERE THE
[5]   ⍝ CHARACTER GROUP IS A VALID NUMBER.  NUMBERS
[6]   ⍝ OUTSIDE RANGE  (⌈/⍳0)<A<⌊/⍳0  COUNTED AS VALID.
[7]    R←' 1111111111234'[' 0123456789.¯'⍳'0 ',A]
[8]    R←1+≠((R∈'23')∨R≠¯1+' ',R)/R
[9]    R←R∈1 12 121 21 31 312 3121 321
       ∇
```

Line 8 of the function uses the not-equal negative difference operation to remove duplicates.

Algorithm Number 146 by Jeffrey Multack, in the September 1980 issue of Quote Quad (Vol 11 No 1) converts numeric input:

```
      ∇R←FI A;M
[1]   ⍝ VERIFY AND CONVERT NUMERIC INPUT.
[2]   ⍝ <A> IS A VECTOR CONTAINING GROUPS OF CHARACTERS
[3]   ⍝ DELIMITED BY ONE OR MORE SPACES.   <R> IS A
[4]   ⍝ NUMERIC VECTOR WITH VALID NUMBERS IN <A> OR
[5]   ⍝ ZERO FOR ANY GROUP WHICH IS NOT A NUMBER.
[6]    R←VI' ',A
[7]    →(∨/R)⍴0
[8]   ⍝ FORM MASK FOR VALID CHARACTER GROUPS.
[9]    M←A≠' '
[10]   M←M>¯1⌽0,M
[11]   M←≠\M\R≠¯1⌽0,R
[12]   R[R/⍳⍴R]←⍎M/A
       ∇

      FI'123 1.3  3¯4  3.4 3E4  ¯3E¯34 E3'
123 1.3 0 3.4 30000 ¯3E¯34 0
```

Lines 10 and 11 have been altered slightly so that they are in the same form as the difference operations already given. Line 10 is the greater-than difference operation which flags the first character in each group. Line 11 then extends the ones for each character group that is valid.

```
      A←'1234  1..23    460'
      ⎕←M←A≠' '
1 1 1 1 0 0 1 1 1 1 1 0 0 0 0 1 1 1
      ⎕←M←M>¯1↓0,M
1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0
      ⎕←R←VI A
1 0 1
      ⎕←M←≠\M\R≠¯1↓0,R
1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1
```

Line 11 can be broken down into 3 stages, a not-equal difference operations:

```
      R≠¯1↓0,R
1 1 1
```

an expansion:

```
      M\R≠¯1↓0,R
1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0
```

and a not-equal scan:

```
      ≠\M\R≠¯1↓0,R
1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1
```

Not equals scan has the property of switching from 1 to 0 and from 0 to 1 every time a 1 is encountered in the vector. The not-equals difference operation does the opposite, so the three stages are: apply a transformation, expand, then put it back to what it was. The partitioned plus reduction does a similar task:

```
R←R-¯1↓0,R←P/+\A
```

Apply a transformation (plus scan), compress, then put it back using the minus negative difference operation. The conversion back to the original form is possible because the minus negative difference operation is the inverse of plus scan.

```
      ⎕←A←+\2 4 1 5
2 6 7 12
      A-¯1↓0,A
2 4 1 5
```

Also, the not-equals difference operation is the inverse of not-equals scan:

```
      ⎕←A←≠\1 1 0 0 1
1 0 0 0 1
      A≠¯1↓0,A
1 1 0 0 1
```

Line 11 of FI is so useful that it should be defined as a function:

```
      ∇R←P PΔMASK A
[1]   ⍝ <P> IS A LEADING PARTITION VECTOR. <A> IS A BOOLEAN
[2]   ⍝ VECTOR WHERE ⍴A IS IDENTICAL TO +/P.
[3]   ⍝ <R> IS A BOOLEAN VECTOR WITH 1 IN EACH PARTITION
[4]   ⍝ WHERE <A> IS 1.
[5]    R←≠\P\A≠¯1↓0,A
      ∇

      1 0 0 0 1 0 1 0 0 PΔMASK 1 0 1
1 1 1 1 0 0 1 1 1
```

Having seen that the basic process is difference, expand, scan, with not-equals, an equivalent function can be written using the same basic process, but with minus and plus:

```
      ∇R←P PΔREPLICATE A
[1]   ⍝ <P> IS A LEADING PARTITION VECTOR.  <A> IS A
[2]   ⍝ NUMERIC VECTOR WHERE ⍴A IS IDENTICAL TO +/P.
[3]   ⍝ <R> IS A NUMERIC VECTOR WITH EACH ELEMENT OF
[4]   ⍝ <A> REPLICATED IN EACH PARTITION.
[5]    R←+\P\A-¯1↓0,A
      ∇

      1 0 0 0 1 0 1 0 0 PΔREPLICATE 2 6 3.2
2 2 2 2 6 6 3.2 3.2 3.2
```

Using the partitioned plus reduction technique of difference, compress, scan, but with not-equals in place of minus and plus, yields a partitioned not-equals reduction function:

```
      ∇R←P PΔNEΔRED A
[1]   ⍝ <P> IS A LEADING PARTITION VECTOR.   <A> IS A
[2]   ⍝ BOOLEAN VECTOR.  <R> IS ≠/ FOR EACH PARTITION.
[3]    R←(1⌽P)/≠\A
[4]    R←R≠¯1↓0,R
      ∇
```

This function can be used to flag partitions with an uneven number of occurrences.

Another 'tool-box' function that illustrates a difference operation is one for removing surplus spaces:

```
      ∇R←SQUEEZE A;B
[1]   ⍝ REMOVE LEADING, TRAILING AND DUPLICATE SPACES
[2]   ⍝ FROM CHARACTER VECTOR <A>
[3]    R←A,' '
[4]    B←' '≠R
[5]    B←B∨¯1↓0,B
[6]    R←¯1↓B/R
       ∇
```

Line 5 has an 'or' negative difference operation which adds a 1 after each group of ones. Line 3 guarantees a trailing space which is then removed on line 6.

The following function is one of a set of functions to help formatting numeric data in VS APL:

```
      ∇R←BRACKETS A;B;C;D
[1]   ⍝ <A> IS A CHARACTER ARRAY OF FORMATTED NUMBERS
[2]   ⍝ WITH AT LEAST ONE SPACE BEFORE EACH NUMBER.
[3]   ⍝ <R> HAS THE NUMBERS MOVED ONE SPACE TO THE LEFT
[4]   ⍝ AND HAS BRACKETS IN PLACE OF NEGATIVE SIGNS.
[5]    R←1⌽,A
[6]    B←R≠' '
[7]    C←B>¯1↓0,B
[8]    D←'¯'=C/R
[9]    R[(C\D)/⍳⍴R]←'('
[10]   C←B<¯1↓0,B
[11]   R[(C\D)/⍳⍴R]←')'
[12]   R←(⍴A)⍴R
       ∇

      NUMS
        .00    368.55      ¯.35     ¯40.10      ¯.34
        .00       .00      6.43    ¯536.58      ¯.04
        .00       .00       .00    ¯761.24       .42

      BRACKETS NUMS
      .00    368.55      (.35)    (40.10)     (.34)
      .00       .00      6.43     536.58      (.04)
      .00       .00       .00    (761.24)      .42
```

Line 7 is a greater-than negative difference which flags the beginning of each group of numbers. Line 10 is a less-than negative difference which flags the beginning of each group of spaces. The negative sign is therefore only replaced by both a left and right parenthesis.

The function was created to help develop a generalised formatter for VS APL. In practice, a formatting function would have the format specification available to indicate where the right parenthesis should be placed.

Partition Functions

Partition functions have been given for plus reduction and not-equal reduction, but partition functions are needed to carry out the equivalent of all the reduction and scan operations. The working Memorandum on Boolean Techniques gives a very comprehensive list, but here are two that are most frequently used:

```
      ∇R←P PΔORΔRED A
[1]   ⍝ <P> IS A LEADING PARTITION VECTOR.  <A> IS A BOOLEAN
[2]   ⍝ VECTOR.  <R> IS ∨/ FOR EACH PARTITION OF <A>.
[3]    R←(P∨A)/P
[4]    R←(P/A)≥R/1⌽R
       ∇

      ∇R←P PΔANDΔRED A
[1]   ⍝ <P> IS A LEADING PARTITION VECTOR.  <A> IS A BOOLEAN
[2]   ⍝ VECTOR.  <R> IS ∧/ FOR EACH PARTITION OF <A>.
[3]    R←(P≥A)/P
[4]    R←(P/A)∧R/1⌽R
       ∇
```

These functions are also published in the FinnAPL Idiom Library numbers 491 and 492.

An example of their use is taken from a set of functions to carry out formatting under VS APL:

```
      ∇R←BLANKΔIFΔZERO A;B;C;P
[1]   ⍝ <A> IS A CHARACTER ARRAY OF FORMATTED NUMBERS.
[2]   ⍝ <R> HAS ALL ZERO NUMBERS SET TO SPACES.
[3]    R←A
[4]    →(0∈B←⍴R)/0
[5]    R←,R
[6]    P←R=' '
[7]    P←P>¯1↓0,P
[8]    P[⎕IO]←1
[9]    C←P PΔMASK←~P PΔANDΔRED R∈' 0.'
[10]   R←B⍴C\C/R
       ∇

      NUMS
        .00    368.55      ¯.35     ¯40.10      ¯.34
        .00       .00      6.43     ¯536.58     ¯.04
        .00       .00       .00     ¯761.24      .42

      BLANKΔIFΔZERO NUMS
              368.55      ¯.35     ¯40.10      ¯.34
                          6.43     ¯536.58     ¯.04
                                   ¯761.24      .42
```

Lines 6 to 8 set up a partition vector, and line 9 creates a mask for those partitions that do not have a space, zero or decimal point. Of course, if the format specification is available the partition vector can be set up without searching the data.

Finally, an example of using partition functions to eliminate looping. In the last issue of the APL User Group News Letter Dick Bowman gave a very interesting problem of calculating geometric means of sets of data.

The solution published calculated the geometric mean of each set of data in a loop, which would be inefficient if large amounts of data were involved and Dick ends his article with 'There surely must be a better way'. The partitioned data approach would be to create a partitioned geometric mean function.

```
      ∇R←P PΔGEOM A;B;C;D
[1]   ⍝ <P> IS A PARTITION VECTOR, <A> IS A NUMERIC VECTOR.
[2]   ⍝ <R> IS THE GEOMETRIC MEAN IN EACH PARTITION WHERE
[3]   ⍝ ALL NUMBERS ARE GREATER THAN ZERO, OTHERWISE ¯1.
[4]    B←P PΔANDΔRED A>0
[5]    C←P PΔMASK B
[6]    D←C/P
[7]    R←*(D PΔPLUSΔRED⍤C/A)÷PΔSHAPE D
[8]    R←(B\R)-~B
       ∇

      ∇R←PΔSHAPE P
[1]   ⍝ <P> IS A LEADING PARTITION VECTOR.
[2]   ⍝ <R> IS THE NUMBER OF ELEMENTS IN EACH PARTITION.
[3]    R←(1⌽P)/⍳⍴P
[4]    R←R-¯1↓0,R
       ∇

      1 0 0 0 1 0 1 0 0 PΔGEOM 3 6 8 1 0 2 3 2 1
3.464101615 ¯1 1.817120593
```

Line 4 finds the partitions that need to be processed, and line 5 creates a mask. Line 7 then calculates the geometric mean for each valid partition, and line 8 sets invalid partitions to negative one. This function can then be used to replace the loop in the original function, after having created a partition vector.

Conclusion

Difference operation and partition functions provide a useful set of tools which help to solve the programming problems where the data does not fit in with rectangular nature of APL arrays.

The generalised array facilities in APL2, NARS and Sharp APL developments provide much more powerful tools for manipulating non rectangular data. The partitioned data approach is much easier and more direct with generalised arrays; the partitioned plus reduction is merely a plus reduction for each element of a vector of vectors, and a proper notation is provided for its application. Roll on generalised arrays — but in the meantime we can make do quite successfully with partitioned functions!

APPENDIX

Boolean Difference Operations

Less-than Negative Difference. The first of each group of zeros is set to one, all other elements are set to zero.

```
      □←A←0 0 1 1 1 1 0 0 1 1 1 0 0 0 1 0 1
0 0 1 1 1 1 0 0 1 1 1 0 0 0 1 0 1
      A<¯1↓0,A
0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0
```

Less-than Positive Difference. The last of each group of zeros is set to one, all other elements are set to zero.

```
      A
0 0 1 1 1 1 0 0 1 1 1 0 0 0 1 0 1
      A<1↓A,0
0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0
```

Less-than-or-equal Negative Difference. The first of each group of ones is set to zero, all other elements are set to one.

```
      A
0 0 1 1 1 1 0 0 1 1 1 0 0 0 1 0 1
      A≤¯1↓1,A
1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0
```

Less-than-or-equal Positive Difference. The last of each group of ones is set to zero, all other elements are set to one.

```
      A
0 0 1 1 1 1 0 0 1 1 1 0 0 0 1 0 1
      A≤1↓A,1
1 1 1 1 1 0 1 1 1 1 0 1 1 1 0 1 1
```

Equal Negative Difference. The first of each group of zeros and the element to the right of each group of zeros is set to zero, all other elements are set to one.

```
      A
0 0 1 1 1 1 0 0 1 1 1 0 0 0 1 0 1
      A=¯1↓1,A
0 1 0 1 1 1 0 1 0 1 1 0 1 1 0 0 0
```

Equal Positive Difference. The element to the left of each group of zeros and the last of each group of zeros is set to zero, all other elements are set to one.

```
      A
0 0 1 1 1 1 0 0 1 1 1 0 0 0 1 0 1
      A=1↓A,1
1 0 1 1 1 0 1 0 1 1 0 1 1 0 0 0 1
```

Greater-than-or-equal Negative Difference. The first of each group of zeros is set to zero, all other elements are set to one.

```
      A
0 0 1 1 1 1 0 0 1 1 1 0 0 0 1 0 1
      A≥¯1↓1,A
0 1 1 1 1 1 0 1 1 1 1 0 1 1 1 0 1
```

Greater-than-or-equal Positive Difference. The last of each group of zeros is set to zero, all other elements are set to one.

```
      A
0 0 1 1 1 1 0 0 1 1 1 0 0 0 1 0 1
      A≥1↓A,1
1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 1
```

Greater-than Negative Difference. The first of each group of one is set to one, all other elements are set to zero.

```
      A
0 0 1 1 1 1 0 0 1 1 1 0 0 0 1 0 1
      A>¯1↓0,A
0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 1
```

Greater-than Positive Difference. The last of each group of ones is set to one, all other elements are set to zeros.

```
      A
0 0 1 1 1 1 0 0 1 1 1 0 0 0 1 0 1
      A>1↓A,0
0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1
```

Not-equal Negative Difference. The first of each group of ones and the element to the right of each group of ones is set to one, all other elements are set to zero.

```
      A
0 0 1 1 1 1 0 0 1 1 1 0 0 0 1 0 1
      A≠¯1↓0,A
0 0 1 0 0 0 1 0 1 0 0 1 0 0 1 1 1
```

Not-equal Positive Difference. The element to the left of each group of ones and the last of each group of ones is set to one, all other elements are set to zero.

```
      A
0 0 1 1 1 1 0 0 1 1 1 0 0 0 1 0 1
      A≠1↓A,0
0 1 0 0 0 1 0 1 0 0 1 0 0 1 1 1 1
```

Or Negative Difference. The element to the right of each group of ones is set to one, all other elements are unaltered.

```
      A
0 0 1 1 1 1 0 0 1 1 1 0 0 0 1 0 1
      A∨¯1↓0,A
0 0 1 1 1 1 1 0 1 1 1 1 0 0 1 1 1
```

Or Positive Difference. The element to the left of each group of ones is set to one, all other elements are unaltered.

```
      A
0 0 1 1 1 1 0 0 1 1 1 0 0 0 1 0 1
      A∨1↓A,0
0 1 1 1 1 1 0 1 1 1 1 0 0 1 1 1 1
```

And Negative Difference. The element to the right of each group of zeros is set to zero, all other elements are unaltered.

```
      A
0 0 1 1 1 1 0 0 1 1 1 0 0 0 1 0 1
      A∧¯1↓1,A
0 0 0 1 1 1 0 0 0 1 1 0 0 0 0 0 0
```

And Positive Difference. The element to the left of each group of zeros is set to zero, all other elements are unaltered.

```
      A
0 0 1 1 1 1 0 0 1 1 1 0 0 0 1 0 1
      A∧1↓A,1
0 0 1 1 1 0 0 0 1 1 0 0 0 0 0 0 1
```

financial & mathematical modellers

APL

books

communications

courses

COMPLETE APL SYSTEM

£2995

includes APL Printer, 256K RAM etc
— all you need.

interpreters

'unusual' software

networks

Alan Pearman Limited
Maple House
Mortlake Crescent
Chester CH3 5UR

0244
46024

IBM
Personal Computer

IBM Authorised Dealer

INSIDE THE INTERNATIONAL APL STANDARD

by David Ziemann

Introduction

This paper constitutes a technical exploration of the contents of the proposed international APL standard. Before the contents are examined, an introduction followed by short sections on the reasons for standardisation and a brief history of the standard are provided. The important concepts of the "conforming implementation" and the "conforming program" are then explained in detail. A description of the features included in the standard is then presented, with a list of some of the features explicitly not included. Finally, a concluding passage discusses ways in which the standard will be of use to the authors of APL implementations and programs.

Casual readers be warned — this paper is a précis of the draft standard and is heavy going.

Why standardise?

APL has been around for 20 years without a standard, so why bother now? Well, the answer lies in the growth of APL, both in terms of the number of establishments using it and the diversification of the language itself. As the number of people using APL grows, communicating between them becomes more difficult, and the need for a definitive APL implementation increases. At a time when there are so many differing implementations of APL on the market, it is important for APL people to come together and see if they can agree upon what constitutes a minimal APL. Only once this is done can we start to exchange APL programs and skills on a global scale. The absence of a suitable standard has doubtless prevented the adoption of APL in many areas, in particular, government departments in the US will not sanction any computer language unless it has a standard.

The purpose of the standard is probably best summarised by quoting from the standard itself which states that "This International Standard defines the programming language of APL and the environment in which APL programs are executed. Its purpose is to facilitate interchange and promote portability of APL programs and programming skills among data processing systems."

A Brief History of the Standard

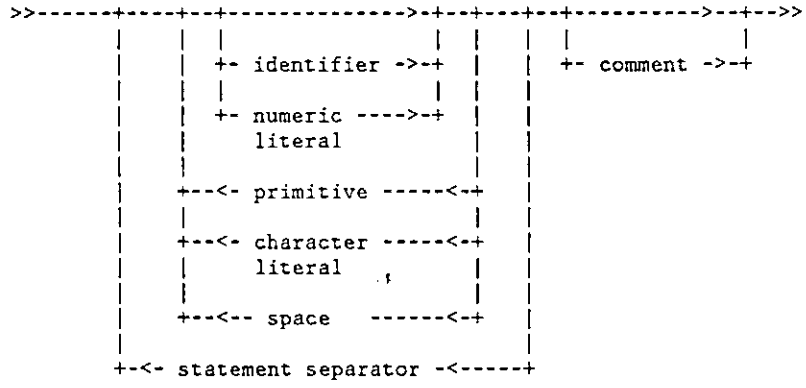
International Standards are produced under the control of the International Standards Organisation (ISO), which works under the auspices of the United Nations and is responsible for producing standards of all kinds. Working through international technical committees, drafts of international standards slowly come into existence and are then voted to become ISO standards by the national standards organisations.

The national standards organisations gather national experts into committees which contribute to the development of national and international standards, submitting their work to ISO or the IEC (International Electrotechnical Commission). They also vote on proposals from ISO (or IEC) for adoption of international standards, and adopt either ISO standards or their own home-grown product as national standards. If a country produces its own national standard before an equivalent ISO standard exists, it usually submits it to ISO as a draft ISO standard.

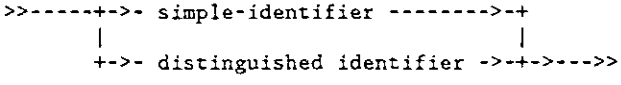
Creating international standards takes a long time, and a lot of forests. Even once agreement has been reached in the international forum (which does occasionally happen), there is still the liaison between the ISO committee and the various national committees and other international committees to contend with. The extensive ISO procedures also have to be followed before the ISO standard is adopted. It can, and usually does, take years.

Figure 1. Some Syntax Diagrams

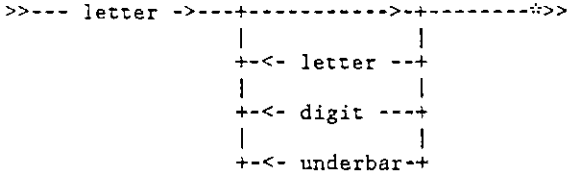
Line



Identifier



Simple-Identifier



In 1979, AFNOR the French national standards body proposed to the ISO technical committee on information processing (TC97), that an International Standard for APL should be produced. The proposal was accepted, and the work was assigned to TC97's sub-committee on programming languages, SC5, which set up Working Group 6 to assist. The first draft appeared in 1980. At approximately the same time, ANSI decided to produce a US APL standard, and so another draft appeared. Needless to say, the two drafts were quite different.

In 1981 it was agreed at an international meeting in the USA, that the two drafts should be merged into a single international working draft. This work proceeded, with contributing experts from the USA, Canada, Japan, Britain, France and other European countries. By August 1983, the fifth working draft had been distributed and accepted by SC5 as the first draft proposal. By the beginning of 1984 the draft proposal was circulated to the SC5 P-members (BSI in the case of Britain) for a three month comment period. These comments will be processed at the next Working Group 6 meeting in Helsinki just before APL84. The result of this will be an instruction to the editor to prepare a second draft proposal. Subsequent processing will then result in an international standard — hopefully, in the not too distant future. When it does, APL will be the first language for which the international standard has preceded any national standard.

This paper was written at the time of the first draft proposal and it is to this document that the remainder of the paper refers.

The Form of the Standard

The standard describes the behaviour of a hypothetical APL machine by specifying how it responds to inputs. The syntax (rules for combining the funny symbols) of this machine is defined by syntax diagrams (or railroad tracks, as they are sometimes called), which are generally easier to understand than the more traditional Backus-Naur form. The standard described three main processes which perform the analysis of syntax:

- line evaluation: this uses a set of syntax diagrams to decompose a line of characters into a list of lexical tokens, working from left to right.
- statement evaluation: this uses syntax diagrams to transform a list of lexical tokens into a list of syntactic tokens, still working from left to right.
- statement reduction: now working from right to left, this uses a phrase table to decompose a list of syntactic units into shorter lists, called phrases, each of which is then evaluated by one of 12 phrase evaluators.

The semantics of APL (what the funny symbols mean) are defined by the behaviour of these phrase evaluators, and that of the formal procedures which they in turn call. These procedures, or evaluation sequences as they are called, are expressed in a formal language which uses English words in a sense precisely defined in the standard. The result is a precise but nevertheless readable document. There is no requirement for an implementation to follow the algorithms suggested in the evaluation sequences exactly, provided that the system produces results indistinguishable from those of the APL machine described in the standard.

As well as specifying the syntax and semantics of APL programs, the standard also specifies the characteristics of the environment in which APL programs are executed, and the requirements for conformance with the standard. In this context, the term 'program' is used broadly, to include everything from a single APL expression to a collection of workspaces communicating via shared variables.

On the other hand, the standard DOES NOT specify:

- required values for implementation limits such as APL workspace site or numeric precision.
- the data structure used to represent APL objects.
- the facilities available through shared variables.

Some Technical Terms

Of fundamental importance to an understanding of the standard are the two terms 'conforming implementation' and 'conforming program'. Loosely speaking, a conforming implementation is an APL interpreter which follows the rules laid down in the standard, and a conforming program is an APL program which would run successfully on the APL machine specified in the standard. In order to define these terms more strictly, we have first to consider the four classes of facility recognised by the standard. These are the defined facility, implementation defined facility, optional facility and the consistent extension.

Defined facility — this is a facility that is fully specified by the standard, and not designated optional or implementation defined. The transpose function, for example, is a defined facility in the standard.

Optional facility — this is a facility that is also fully specified by the standard, but is designated optional, as a conforming implementation may or may not include it. The APL statement separator is an example of an optional facility.

Implementation defined facility — this is a facility that is NOT fully specified by the standard, and is designated implemention defined. The algorithm used by an implementation to generate pseudo-random numbers, for example, is an implementation defined facility, but the APL functions roll and deal are defined facilities and must be provided as specified.

Consistent extension — this is a facility that is NOT specified at all by the standard, but if included in an APL implementation will not give rise to an error in circumstances in which the APL machine would. In other words, an error report arising from a specific input to the APL machine in the standard can be replaced by some other behaviour in the actual implementation. For example, you can bring out an implementation with a brand new APL function in it (say a dyadic execute function) because this would replace an error report by some other action. The consistent extension is the only mechanism by which extra APL features may be added to an implementation.

Conforming Implementations

We can now look more closely at what a conforming APL implementation actually is. In order to conform to the standard, an implementation must provide all the defined facilities and implementation defined facilities specified in the standard, and each defined facility must behave exactly as specified. Additionally, a conforming implementation may provide any or all of the facilities described in the standard as optional. If included, the facility must behave exactly as specified. Finally, a conforming implementation may also include any consistent extensions. Because the consistent extention mechanism is the only way the standard permits an implementation to provide extra APL features, the standard itself tends to adopt a 'minimal' approach, including only minimum agreed features. Where implementations differ, the standard will often require an error. This is not meant to encourage an implementation to produce an error under the particular circumstance, but to allow more than one interpretation to be standard conforming. Some examples of this will be described later. The observant reader may have noticed one loophole that needs to be closed here. A program that attempts to use an optional facility that is not provided in a particular implementation will generate an error. It is therefore NOT permissible for an implementation to replace such error signalling by any other behaviour.

Implementation algorithms and implementation parameters

Before we proceed, it will be necessary to look at two further concepts — implementation algorithms and implementation parameters. Facilities in the standard are described both informally in English and procedurally using what are called evaluation sequences. These evaluation sequences often refer to algorithms not defined in the standard, and whose behaviour is deemed

implementation defined — that is, it's up to the implementor to provide an exact definition. These implementation algorithms fall into five groups:

The implementation algorithms

- Pythagorean algorithms; eg sine, inverse cosine, hyperbolic tangent
- General numeric algorithms; Exponential, gamma function, modulo, natural logarithm and power
- Seminumeric algorithms; the pseudorandom number generator, and the algorithm for the deal function
- Linear algebraic algorithms; this group only contains the algorithm for generalised matrix divide
- System dependent algorithms; eg how numeric literals are converted to internal numbers, how the current time is generated, how a function trace display is to be presented, and others.

Each one of these algorithms is referred to by the standard, but must be defined by the particular APL implementation. In addition to these algorithms are the implementation parameters. These are quantities referred to by the standard but whose values are implementation defined. Over twenty of these are recognised by the standard, and some examples follow:

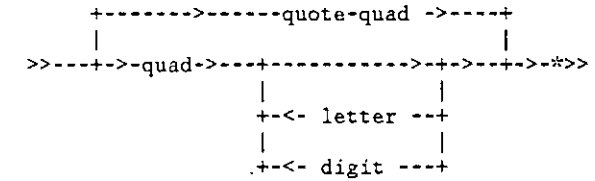
Some implementation parameters

- Atomic vector: an implementation defined character vector containing every element of the character set exactly once.
- Positive number limit: the number (in machine rather than mathematical terms) greater than all other numbers.
- Rank limit: an integer specifying the maximum value for the rank of an array. The limit must apply uniformly to all arguments and results of primitive operations.
- Identifier length limit: an integer specifying the maximum number of characters in an identifier.
- Comparison tolerance limit: the largest value permitted by the implementation for the system variable quadCT.
- Integer tolerance: a value used to determine whether a given number is to be considered integral or not.

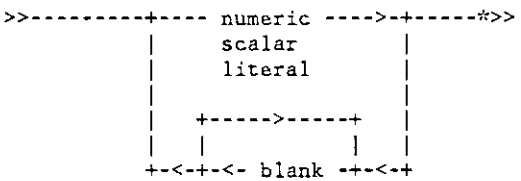
The question then naturally arises — what happens if some action causes a limit specified by one of these implementation parameters to be exceeded? For example, suppose you try to increase the rank of an array which already has the maximum number of dimensions permitted by the particular implementation you are using. Clearly, it would be nice if the implementation complained about this, rather than just ignoring the attempted action, or even taking some other action and not telling you what it's done. (A good example of the latter is the behaviour of some implementations when you try to reference a variable whose name contains more characters than the identifier length limit — the name is very often merely truncated to the maximum permitted length, and you don't even get to know about it!)

Figure 2. More Syntax Diagrams

Distinguished-Identifier

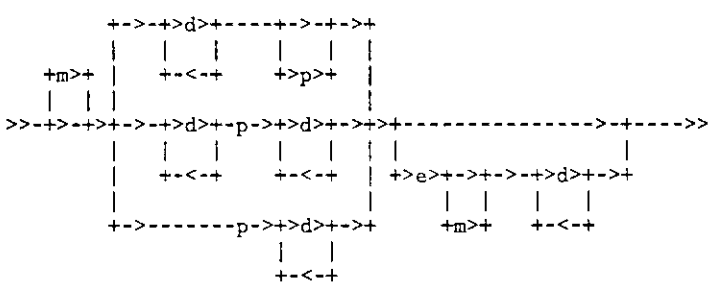


Numeric-Literal



Numeric-Scalar-Literal

d stands for **digit**.
e stands for **exponent-marker**.
m stands for **overbar**.
p stands for **dot**.




```
* Example:
*
*    ¯12.34567E¯890
```

The problem is solved by requiring a conforming implementation to signal an error if any action is taken that would result in an implementation parameter limit being exceeded. But what error? Out of the existing popular error messages only DOMAIN ERROR comes close, but not close enough. A DOMAIN ERROR should strictly only be reported when a function argument lies outside the domain of the function — it is an abstract mathematical concept rather than an indication of a shortcoming of a particular implementation of the APL language. So, a small bit of creativity was experienced here, and the standard requires that a new error message, LIMIT ERROR, be signalled in such cases.

Required documentation for conforming implementations

A conforming implementation must provide documentation relating to its optional facilities, implementation defined facilities and consistent extensions. It must document the presence or absence of each of the facilities described in the standard as optional, and also the following aspects of the implementation defined facilities:

— a description of the character set. This must include a chart showing the correspondence between the atomic vector and the characters in the required character set. (The required character set is a standard-specified subset of an implementation defined finite set of characters called the character set).

— a description of the numbers, including a characterisation of the internal representation used. (Note that the numbers are an implementation defined finite set whose elements are used to represent arithmetic quantities).

— descriptions of the characteristics of each implementation algorithm.

— the value of each implementation parameter.

As far as consistent extensions goes, each one provided must be documented by a conforming implementation. The documentation must also clearly state that use of a consistent extension prevents a program from conforming with the standard.

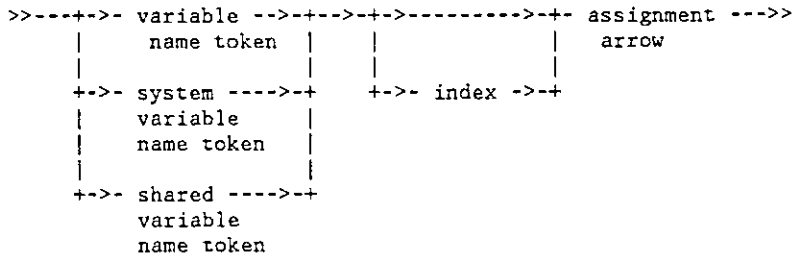
This documentation has often been colloquially referred to as the 'toaster plate' of an implementation because, like a plate on an electrical appliance, it reveals to the potential purchaser the salient features and limitations of the product.

People occasionally ask why implementation parameter limits like the rank limit and identifier length limit are not hard-coded into the standard. To do this means deciding upon a minimum value necessary to achieve conformance, and this is hard to do, as everyone has different ideas about what constitutes a minimal APL. For example, what would you choose as the minimum allowable rank limit for an APL implementation? Typical answers are 3, 8, 15, 63 etc. These answers are arbitrary, and often have more to do with people's ideas of how computers work than with useful limits. Let's say we decide on 15 as our minimum rank limit. This means that we reject as non-conforming any APL implementation that allows no more than 14 dimensions in any array. Alternatively, we may encourage an implementor to use a more inefficient storage representation for arrays just because we insist upon a rank limit of 15 rather than 14. The approach in the standard has therefore been, loosely speaking, that implementors can decide the values of these limits themselves, but they've got to tell everyone loud and clear what they are.

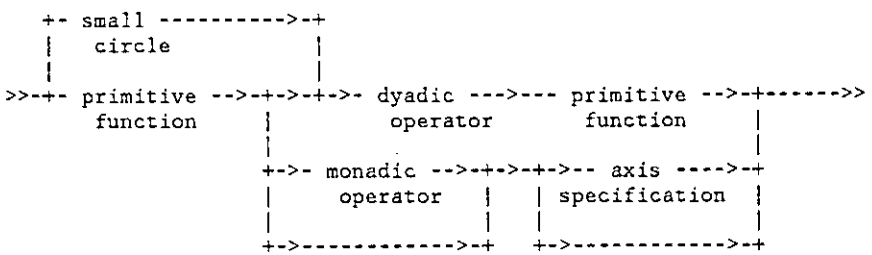
If you see two conforming implementations, one with a rank limit of 3 and one with a rank limit of 30, you will know which one to buy, just as you know which electric fire will warm you if one is rated at 3 watts and the other at 3 kilowatts. The attitude is very much one of 'let the market decide'.

Figure 3. Yet More Syntax Diagrams

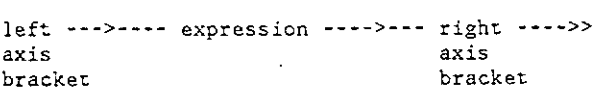
Assignment



Derived-Function



Axis-Specification



Conforming Programs

We now know how to spot a standard-conforming APL implementation, but what about standard conforming APL programs? It is extremely important to be able to write a conforming program if you want to port it between conforming implementations.

A conforming program can use only facilities that are specified in the standard, that is defined, implementation defined and optional facilities. A conforming program cannot use consistent extensions. Also, a conforming program cannot depend on the signalling of any error by a conforming implementation. This is because consistent extensions that replace errors are permitted in conforming implementations. This means that a conforming program cannot use an error trapping facility (which may be provided as a consistent extension by a conforming implementation), and more interestingly, the standard in its present form can never include one as a defined, implementation defined, or even optional facility.

Note that in general, the presence of a consistent extension in a conforming implementation shall not affect the behaviour of a conforming program. Remember also, that errors produced by the absence of an optional facility cannot be replaced by consistent extensions in a conforming implementation, since this would affect the behaviour of conforming programs that use the optional facility.

Implementors of conforming implementations are also discouraged from replacing LIMIT ERRORS with consistent extensions, since these errors are the only safeguards a conforming program has when attempting to operate in a conforming implementation whose implementation parameters are inadequate to support it. For example, if the LIMIT ERROR on the identifier length limit were not signalled, a conforming program with identifiers longer than the local identifier length limit would malfunction without warning. Currently the standard only warns implementations off doing this, but does not prohibit it.

Required documentation for conforming programs

A conforming program must document which of the optional features described in the standard it requires. It also has to document any specific minimal values required for implementation parameters. For example, it may perform calculations upon numbers as large as 1E100 and would therefore require an implementation whose positive number limit was at least this large. The document should also state, for another example, the length of the longest identifier name in order to determine whether the program can run on a particular implementation. Generally, the requirement for each of the implementation parameters should be documented in order to determine the program's suitability for a given conforming implementation.

It is not surprising that a non-conforming program can produce unexpected results when run on a conforming implementation, but more surprising that the same is true of conforming programs. In fact, a conforming program may or may not work, and may or may not produce identical results on different conforming implementations, due to inherent dependencies on implementation parameters or implementation algorithms. For example, the algorithm used for matrix divide is implementation defined, and may or may not generate a DOMAIN ERROR for given arguments on different conforming implementations. Another example is the implementation parameter called integer tolerance. The value of this parameter is used by an implementation to determine whether a given number is to be considered integral or not. So the same index expression run on two conforming implementations could produce a result on one and an error on the other, dependent on their respective values for this parameter. This situation seems unavoidable, but is alleviated when conforming programs document required values for such parameters.

Figure 4. Some Phrase Evaluators

```
    5.3.12  And

            Z ← A ∧ B

*       Z is the Boolean product of A and B.

    Evaluation Sequence:

            If either A or B is not near-Boolean, signal domain-error.

            Set A1 to the integer-nearest-to A.
            Set B1 to the integer-nearest-to B.
            If either A1 or B1 is zero, return zero.
            Otherwise, return one.


*   Example:
*
*               0 1 ∘.∧ 0 1
*     0 0
*     0 1



    5.3.13  Or

            Z ← A ∨ B

*       Z is the Boolean sum of A and B.

    Evaluation Sequence:

            If either A or B is not near-Boolean, signal domain-error.
            Set A1 to the integer-nearest-to A.
            Set B1 to the integer-nearest-to B.
            If either A1 or B1 is one, return one.
            Otherwise, return zero.


*   Example:
*
*               0 1 ∘.∨ 0 1
*     0 1
*     1 1
```

APL facilities in the standard

In this section we will summarise the content of the standard in terms of the APL features included within it. First let's take a high-level view of what has been included in and excluded from the standard. This will be done by comparison with IBM's VS APL, because VS APL is widely known, and is most similar to the APL machine described in the standard.

The standardisation committee agreed early on in the process that the standard should describe existing practice in APL rather than establish a new language level. As a guideline, a feature is considered for standardisation if it exists in at least two existing APL implementations. In other words, the standard documents rather than invents, as far as possible. (In fact, there are some minor exceptions to this as we shall see).

Here is a list of defined facilities included in the standard:

- all of the primitive functions and operators from VS APL that you have come to know and love.
- the system functions quadTS, quadAV, quadLC, quadDL, quadNC, quadEX, quadFX and quadCR.
- the system variables quadCT, quadRL, quadPP, quadIO and quadLX.
- quad and quote quad input and output.
- entry and editing of niladic, monadic and dyadic user defined functions via a minimal del-editor.
- the system commands)CLEAR,)COPY,)DROP,)ERASE,)FNS,)LIB,)LOAD,)RESET,)SAVE,)SI,)SINL,)VARS,)WSID.
- the use of the underbar character (_) in all but the initial character of identifiers
- the use of comments to the right of, and on the same line as executable code (end of line comments).

The last two points constitute added features to VS APL.

Currently only three facilities are described as optional in the standard:

- shared variables via the system functions quadSVO, quadSVQ, quadSVC and quadSVR.
- the diamond statement separator.
- trace and stop control via the system functions quadTRACE and quadSTOP.

The standard specifically does not include, or make any reference to, the following facilities:

- a component filing system via system functions.
- trace and stop control via the Tdelta and Sdelta syntax.
- the implementation dependent system functions quadAI, quadWA and quadPW.
- the grouping of identifiers via system commands such as)GROUP,)GRP or)GRPS.
- the creation and behaviour of locked user defined functions.
- mechanisms for creating and handling enclosed or generalised arrays.
- error trapping.
- pass-through localisation, ie. initialisation of localised system variables to the value of their global homonyms.

In fact, all of these facilities (except the Tdelta and Sdelta syntax) could be provided by an implementation as consistent extensions, although a conforming program could not then employ them.

Let's now look at some of the specified facilities in more detail:

Primitive functions

These are as found in VS APL, with no omissions and no additions. There are some interesting points to note however:

- if an argument to a scalar function is empty, then the result is also empty, and has a type dependent on the function being used. For example, the expression "3-" will yield an empty numeric vector, as will the expression "+".
- in the standard, roll and deal are classified as mixed rather than scalar functions because their result arrays cannot be generated in parallel.
- the standard has been designed with a view to allowing complex arithmetic as a consistent extension. This is apparent in two areas:
  i) the more usual definition of the circular function with a left argument of "-4" has been replaced by one which would permit such a consistent extension.
  ii) the distinction made in some implementations of APL between rational and irrational powers has been eliminated to allow complex arithmetic as a consistent extension. For example, the standard requires that "-8" raised to the one-third power should yield a domain error, because the right argument of power is not integral. This behaviour can then, of course, be replaced by a suitable consistent extension.
- the result of "A,B" where A and B are empty vectors is always the value of A. For example, the result of the expression "'',0/0" is the empty character vector, whereas the result of "(0/0),''" is the empty numeric vector. In practice, this seems to differ from system to system, with some giving the result as the left argument, some the right argument and some either the numeric vector always or character vector always. The standard has gone for the left argument, but the point is still under discussion!
- in accordance with the 'minimal' approach, the definition of the base value (decode) function is stricter than on many implementations. It does not require that a unit inner dimension of one argument be replicated to match the corresponding inner dimension of the other. This behaviour can be provided as a consistent extension.
- indexed assignment is defined so that assigning 1 2 3 into A[1 1 1] causes A[1] to have the value 3. This not necessarily obvious result comes from the decision to process the assignments in the ravel order of the index array rather than notionally in parallel. An alternative would have been for the standard to require a domain error, and leave the implementor to provide a suitable consistent extension. It was felt, however, that in this case standardising the result was more useful than ignoring the problem.

System function and system variables

The standardisation committee spent a lot of time and expended much energy in this area, although there are few surprises. The primary difference between system functions and system variables is that an error is signalled if the name of a system function appears in the list of local names of a user defined function header. Among the points of interest are the following:

- quadTS uses an implementation defined facility to generate its result.
- quadLC contains only elements relating to user defined functions, and not to contexts created by the use of the execute function or quad input.
- quadNC and quadEX signal DOMAIN ERROR if their arguments contain names which are not simple identifiers, so that consistent extensions can be made. A simple identifier is an identifier which does not start with a quad or quote quad character.
- the standard ensures that an error is reported if the syntax class of any tokens in a statement changes during execution of that statement. For example, the expression "F quadEX'F'" attempts to do just this, where F is a user defined monadic function. A syntax error would be reported in this case. Similarly, an

attempt to fix a function G by using quadFX in an expression that forms the left argument of a call to the dyadic function G will result in a DOMAIN ERROR. A conforming implementation may relax these restrictions, of course, but a conforming program must abide by them.

— the evaluation sequences for assignment to the system variables report DOMAIN ERROR if the value to be assigned is unacceptable to that system variable. In addition, primitive functions that implicitly use system variables will report IMPLICIT ERROR if the relevant system variable has no value. This can happen if a system variable is localised to a function and not set before it is implicitly referenced by a primitive function. Conforming programs that localise system variables should therefore assign them valid values before calling any primitive operations that require them.

User defined functions

— user defined functions may be created and edited via a minimal del-editor described in the standard. Currently this is a defined facility, and is therefore required by a conforming implementation, but some people feel that in these days of clever full screen editors, it is unreasonable to insist upon a line-editor. It is possible, therefore, that the facility may become optional in a later draft.

— the class of the result name in a user defined function cannot be defined function or shared variable. A value error is signalled under such circumstances in order to prevent conforming programs from causing an exit from a function when the result name has one of these classes.

— an error should not be reported by a conforming implementation if a defined function fails to assign to its result name and the calling context does not require a value.

System commands

— thirteen system commands are currently included as a defined facility, although there is some feeling that these should be made optional, or even removed altogether. Some implementations provide the required operations by way of system functions. In the standard, system commands cannot be entered during function definition mode, or invoked as an argument to the 'execute' primitive.

Shared variables

— shared variables are an optional facility of the standard. They provide an interface between cooperating APL sessions, although they may also be used to provide an interface between APL programs and non-APL system facilities. Conforming programs that use shared variables must document this fact, since shared variables are not a defined facility of the standard.

Trace and stop control

— trace and stop facilities are provided in the standard via the system functions quadTRACE and quadSTOP. The facility is optional because it is only just coming into widespread use. The two functions are the only example of full English words being used in the names of system functions or variables. However, the standards committee were unanimously agreed that the Tdelta and Sdelta syntax would not be cast in concrete, and so some design had to be done in this area. The right argument is always the function name. When used monadically, the result is the line numbers of the lines currently set for trace/stop. Dyadically, the left argument is a vector of line numbers for which a trace/stop set is required. The result is then the line numbers of the prior set. If an attempt to trace or stop line zero of a

Figure 5. More Phrase Evaluators

```
  8.1.7  Execute

         Z ← ⍎ B

*        Z is the result of evaluating the character scalar or vector B
*        as a line of APL.

  Evaluation Sequence:

         If the rank of B is greater-than one, signal rank-error.
         If any item of the ravel-of B is not a character, signal
           domain-error.
         Generate a new context with

              mode set to execute.
              current-line set to the ravel-of B.

         Append the new context to the state-indicator as a new first
           item.
         Set Z to evaluate-line.
         Remove the first context from the state-indicator.
         Return Z.


* Examples:
*
*              ⍎'T←3'
*              T
*     3
*              ⎕←⍎'T←3'
*     3
*              A←⍎''
*     value-error
*

  Note:

  If an error is signalled during execute, the user should be able to
  determine from information provided by the system where the error
  occurred in the argument of execute as well as where the failing
  execute primitive occurred in the immediate-execution or
  defined-function line.
```

function is made, a DOMAIN ERROR is reported, thereby allowing consistent extensions to be made here.

Finally, let's take a quick look at the reasons for excluding certain facilities from the standard. In general, these reasons can be summarised in the following list:

1. The facility has not achieved widespread use among many implementations of APL.
2. The function provided by the facility is fairly widespread, but different APLs provide the feature via a different syntax.
3. The facility is too closely bound to the specific nature of the operating environment rather than to the APL language itself.
4. Inclusion of the facility would result in the APL language being compromised, from a theoretical viewpoint.

These considerations have to be balanced against the commercial desirability and usefulness of a particular feature, and these judgements are of course subjective.

Good arguments against including a component filing system, for instance, can be constructed from all the points on the above list, although this is left as an exercise for the reader.

CONCLUSION

As well as improving the portability of APL programs, the international APL standard will also promote the use of APL itself, and improve the quality of APL implementations. The standard will prove (and already has proven) invaluable for APL implementors who wish to conform to it. It can be used to form the basis of an implementation and to encourage implementors to provide extensions to the language which are consistent with the standard. The standard will also be important for software vendors who wish to write standard conforming programs. It will encourage them to reduce, isolate and document those areas of their packages that are not standard conforming. Finally, and most importantly, those who purchase APL interpreters and APL programs will expect to see the product documentation required by the standard for conforming implementations and programs. The prospective purchaser of an APL interpreter should be able to determine quickly whether it conforms to the standard, and what optional features and consistent extensions it provides. Those buying APL software should likewise be able to determine the suitability of APL programs for their particular installations, and to predict the areas in which problems might arise.

However, the standard is over 300 pages in length and it requires a fair amount of dedication to read and understand. Interpretation of the standard is therefore not a task to undertake casually.

REFERENCES

[1] The First Draft Proposal for the International APL Standard. ISO document number: ISO DP8485 APL.

[2] Standardisation of APL, J.M. Sykes, Chairman BSI APL group. Appearing in Computer Bulletin, March 1984.

[3] Language Standards for APL, P. Barnetson, IBM UK Ltd., September 1983.

BRITISH APL ASSOCIATION

Membership Application Form

Please read the membership information in the inside front cover of VECTOR before completing this form. Existing members 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, N. Staffs Polytechnic, Blackheath Lane, Stafford ST18 0AD, 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.

1984 COMMITTEE

Chairman:	Philip Goacher 01-637 0471	The British Computer Society, 13 Mansfield Street, London W1M 0BD
Secretary:	Anthony Camacho St. Albans 60130	2 Blenheim Road, St. Albans, Herts AL1 4NR.
Treasurer:	Mel Chapman 0785-53511	N. Staffs Polytechnic, Blackheath Lane, Stafford ST18 0AD.
Activities:	Dick Bowman 01-634 7639	CEGB, 85 Park Street, London SE1.
Education:	Chris Beatty	220 Balham High Road, London SW12.
Journal Editor:	Robert Bittlestone	26 Barham Road, London SW20 0ET.
Publicity:	Vacant	
Technical:	Vacant	

ACTIVITIES WORKING GROUP

David Allen
Dick Bowman
Dominic Murphy
David Preedy
Stan Wilkinson

JOURNAL WORKING GROUP

Jonathan Barman
Robert Bittlestone
David Preedy
Adrian Smith
David Ziemann

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" — an 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*Plus Ltd.	Aston Science Park, Love Lane, Birmingham B7 4BJ. Tel. 021-359 5096
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	132 Buckingham Palace Road, London SW1W 9SA. Tel. 01-730 4567

The British Computer Society, 13 Mansfield Street, London W1M 0BD.