Contents

Quick reference diary

2-5 Oct	Boston Massachusetts, USA	Dyalog conference
20-24 Sep	Syracuse University, USA	Minnowbrook conference

Dates for future issues

Vector articles are now published online as soon as they are ready. Issues go to the printers at the end of each quarter – as near as we can manage!

If you have an idea for an article, or would like to place an advertisement in the printed issue, please write to editor@vector.org.uk.

EDITORIAL

I recently returned from visiting Japan for the first time in my life, where I had the pleasure of spending a day with Kyosuke Saigusa of APL Consultants of Japan. He was revelling in the support for Unicode introduced in recent releases of APL2.

Japanese writers nimbly mix a phonetic script, a script based on Chinese ideographs, the Roman alphabet (the *romaji*), and both Arabic and Japanese numerals. These scripts are laid out both left-to-right, top-to-bottom and top-to-bottom, right to left. Japanese typographers compose pages in which these reading directions are mixed, a challenge to both aesthetics and agility.

In this world the elegant glyphs of APL seemed from the very start to fit in very comfortably. When working at IBM years ago, Saigusa-san translated the first APL manual into Japanese. But what typographers in Japan manage as routine was problematic with computers until the advent of Unicode.

Saigusa-san describes in this issue the tutorial tool work he and his son have built to make APL2 more accessible in Japanese.

We're pleased to publish a memoir of Donald McIntyre, for many years a contributor to Vector, written by his friend Keith Smillie. McIntyre was an early adopter of what Ken Iverson pioneered and dubbed as ‘expository programming’. (See *Vector* Vol.22, No.3)

Plenty to study in this issue. Jan Karman gives the next instalment of *Financial Math In q* and Neville Holmes of *Functional Calculation*. Stevan Apter's *No Stinking Loops* has a meaty and cunningly-constructed tutorial from Stevan Apter on “Tables with calculated columns”. And *J-ottings 53* simplifies betting using complex numbers.

Sylvia Camacho profits from J to conduct a “stroppy” dialogue with Graham Parkhouse on certain infinite series. Alan Sykes puts Dyalog APL to work to get a class library for statistics. And John McInturff finds J convenient for constructing odd-order magic squares.

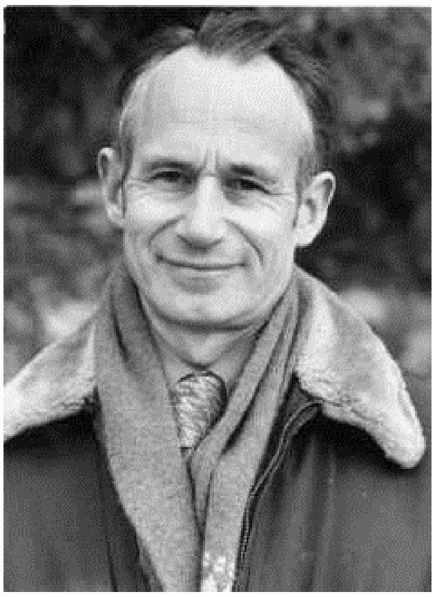
Enjoy.

Stephen Taylor

N E W S

Remembering Donald McIntyre

by Keith Smillie (smillie@cs.ualberta.ca)



Donald McIntyre 1923-2009

Donald McIntyre, geologist, teacher, lifelong APL user and contributor to *Vector*, died in 2009. He is remembered here by his friend Keith Smillie.

The I. P. Sharp Newsletter for November/December 1980 in a brief account of the APL Users Meeting held in Toronto that year has a picture of Donald McIntyre at the podium with the following caption:

Donald B. McIntyre, Pomona College, Claremont, Ca, who has persuaded a liberal arts college, faculty and staff alike, to use APL before all other computer languages (commenting on the concept of the 'empty vector'): "I think we have the Arabs to thank for inventing zero, but I know that we have Dr Iverson to thank for inventing nothing."

Donald's contribution to the conference, in addition to his presence which was always warmly welcomed wherever he went, was a 30-page paper, "APL in a liberal arts college", describing his work in introducing computing in general and APL in particular to Pomona College, which still makes stimulating reading 30 years later.

As readers of *Vector* know, Donald died on 21 October 2009 at the age of 86 after a long illness. It is the purpose of this article to pay tribute to him as a person and to his contributions not only to the APL/J community but also to Scottish geology and to the larger world of scholarship. Some of the material here has been taken from an earlier appreciation of his life and work, "Donald McIntyre: Geologist, historian and array language advocate 1923-2009" which was published in the *Annals of the History of Computing*.

Early life and education

Donald Bertram McIntyre was born on August 15, 1923 in Edinburgh, Scotland. His parents were the Reverend R. E. McIntyre, a Church of Scotland minister, and Mary Darling McIntyre. He was educated at George Watson's Boys College, but during the war he was evacuated to Grantown Grammar School in the Scottish Highlands. While at Grantown he became an enthusiastic mountaineer. One of his earliest adventures was climbing the seven highest Cairngorms within twenty-four hours. Challenged by a published statement in the *Scottish Mountaineering Club Journal* that the six highest peaks – there were seven and the name of the highest had been omitted – had twice been climbed in one day, Donald and a fellow schoolboy climbed all seven in a twenty-hour expedition covering 35 miles and an ascent of 10,300 feet. According to Donald the climb was made difficult at one stage by an "adverse conspiracy of darkness, cloud, aneroid, and lack of previous acquaintance with this section of the traverse".

In 1942 Donald entered the University of Edinburgh and soon "came under the spell", to use the words of one writer, of Arthur Holmes who had just been appointed Regius Professor of Geology. He was one of three students in Holmes's first class in Advanced Physical Geology, and his lecture notes from this class are now in the University's Special Collections. Donald graduated with a B.Sc. in geology in 1945. In the same year he presented his first paper, "The crystal structure of Apatite and its relation to tooth and bone material", written jointly with Arnold Beevers, then Dewar Fellow in Crystallography at the University of Edinburgh. Donald was always very proud of this paper which was presented to the Mineralogical Society and which represented an early contribution to the understanding of the role of fluorine in the development of teeth.

In 1947 Donald received a Ph.D. in geology from the University of Edinburgh with a thesis supervised by Professor Holmes. He spent the following year at the University of Neuchâtel in Switzerland studying plate tectonics under C. E. Wegmann, Professor of Geology. While in Neuchâtel he developed an interest in the history of science which was apparent throughout his career in his lectures and published papers. On his return to Edinburgh in 1948 he took up a three-year appointment as Lecturer in Economic Geology which was followed by another three-year appointment as Lecturer in Petrology, a position he also held for three years. During this period his research interests were in tectonics and the study of the deformed rocks in the Scottish Highlands. In 1951 he received a D.Sc. in geology also from the University of Edinburgh.

Geology

In addition to his field work Donald became a leading authority on the life and work of James Hutton (1726-1797), a leading figure in the Scottish Enlightenment, which Donald has described as "that constellation of friends and kin that gave the world modern philosophy, modern economics and much of modern science". Hutton is known as the "father of modern geology" and was the first to give compelling evidence that the Earth was a million times older than the figure of 6000 years provided by Archbishop James Ussher, who in 1658 from an examination of biblical records said that the Earth was created on the night preceding 23 October 4004 BC. Hutton published his arguments in 1795 in his two-volume *Theory of the Earth* and was working on a third volume at the time of his death two years later. Illustrations intended for publication in this work were lost; Donald played a role in their discovery and publication almost 200 years later.

Donald was the opening speaker in The Royal Society of Edinburgh's bicentennial celebration in 1997 of Hutton's death. His paper "James Hutton's Edinburgh: The historical, social and political background" with its 12 pages of bibliography was later published in the journal *Earth Sciences History*. Earlier that same year on a wet and windy 26 March in Greyfriars Kirkyard in Edinburgh he gave an eulogy to mark the exact bicentennial of Hutton's death. Also in the same year Donald co-authored with Alan McKirdy of the Scottish National Heritage the monograph *James Hutton, The Founder of Modern Geology*, an attractive and beautifully illustrated and written life of James Hutton for the general reader, which was revised in 2001.

One of Donald's fellow graduate students, who was also studying under Professor Holmes, was Ma Hsingyuan, one of the few Chinese students then studying in Britain. He and Donald went on geological and mountaineering trips in Scotland and Switzerland, and he soon became a family friend – and was the dancing partner of Donald's mother on two occasions. He returned to China in 1948, where he disappeared from Donald's view until he unexpectedly called Donald at Pomona College from New York in 1985. He was then Director of the State

Seismological Bureau in Beijing and President of the Geological Society of China. He and Donald were able to renew their long-interrupted friendship both in the United States and in Beijing where Donald was invited to give lectures and workshops on computers and geology. Ma died in 2001 after a long illness.

Pomona College

In 1954 Donald was invited to join the Department of Geology at Pomona College, a small liberal arts college in Claremont, California whose curriculum included astronomy, botany and geology. The geological attractions of California might be best described in his own words: "…whereas Scotland's geological activity was in the distant past, California is geologically young. Imagine the thrill of a scientist seeing a live animal for the first time, having previously been familiar with the detailed anatomy of only dead animals. So it was for a young geologist leaving Scotland for California!" Donald joined the Department of Geology as an Associate Professor. The following year on the retirement of the long-time Chairman Professor A. O. Woodford he became Chairman, a position he held until 1984. In 1957 he was promoted to Professor, and in 1986 was appointed Seaver Professor of Science.

First steps in computing

Donald's use of computers in geology began with what he termed a "state-of-the-art Marchant electronic calculator". In April 1964 Pomona College ordered an

IBM System/360 Model 40, which was installed in September 1965. Donald made use of this system in his own work and in informal classes in Assembler and Fortran. His first introduction to array languages was given by the formal description of the System/360 in what was then called Iverson Notation which appeared in the *IBM Systems Journal*, vol. 3, nos 2 and 3. He later said that he “pored over this remarkable document for years”. He made use of the notation in what was the first credit course in computer science at Pomona College in the 1968/69 academic year. This course and Donald’s subsequent very extensive work in APL and J will be described in the following section.

In 2001 Donald wrote an account entitled “My Involvement in the Use of Computers” which described his work with computers at Pomona College. Today it makes fascinating reading with its descriptions of the computing technologies available from the 1950s to the 1980s and the methods, most of them primitive by today’s standards, of their use. Near the end of the paper he remarked that in preparing this account he was “struck with the number of chance occurrences that have played so great a role in determining my professional life”, and that he kept thinking of the following lines from Robert Frost’s “The Road Not Taken”:

Two roads diverged in a wood,
and I –I took the one less travelled by,
and that has made all the difference.

Perhaps all of us in the APL/J community by choosing these languages have taken “the one less travelled by”, and our lives are the richer for it.

During his time at Pomona College Donald gave a very large number of banquet addresses, invited papers and lectures, and workshops in the United States, Canada, Great Britain and other countries on a variety of topics in geology, computing and the history of science. He was twice appointed National Lecturer for the Association of Computing Machinery. In 1985 he was named California Professor of the Year. In the same year he received the medal of the Geological Society of China. In 1986 he was appointed a Distinguished Fellow of the University of Edinburgh. Most fittingly in 1994 he received the Kenneth E. Iverson Award for Outstanding Contributions to the Development and Application of APL.

He gave the Convocation address, “Footprints on the Sands of Time”, when Pomona celebrated its centennial in 1987. In this address, which fortunately has been preserved as an MP3 file, Donald used his extensive knowledge of the history of science and his skills as an expositor to place the Earth, its inhabitants and Pomona College in their rightful places in the cosmos. Hearing this address today is as moving as it obviously was to the Convocation audience when it was first given.

In spite of a busy and productive academic and professional life Donald did not neglect his personal life for in December 1957 he returned to Scotland and married Ann Alexander of Edinburgh in a ceremony conducted by Donald's father. Among the guests were Professor Wegmann, with whom Donald had worked in his postdoctoral year in Neuchâtel, and Mrs Wegmann. The Wegmanns later visited Donald and Ann on their honeymoon in the Highlands. Donald and Ann's marriage was welcomed by some of his students who felt that he “needed a wife to feed him well and bring out his shy sense of humor” and were “pleased to hear he'd brought one from Scotland, surely a rosy-cheeked and capable woman!”.

Donald and Ann had one son, Ewen, who was named after Professor Wegmann. As Ewen is disabled with cerebral palsy, he always has required special care. Donald always took an active part in this with some of his care reflecting his own many interests. When Ewen was a little child, Donald would often play him lullabies on the bagpipe chanter. Later Donald wrote a suite of personalized computer programs which enabled Ewen to have some employment in a local California hospital. Ewen even participated in some of his father's mountaineering activities, and on one occasion Donald pushed him halfway up the Matterhorn in his wheelchair!

Donald's contributions to Pomona College might best be summed up in the eulogy by David Alexander who was Pomona's President from 1969 to 1991:

In the history of Pomona College Donald McIntyre stands among the titans of its leadership. His uncountable contributions to the life and program of the College as a member of the faculty from 1954 to his retirement in 1989 admit him to the company of those heroic persons who have created and sustained the College's excellence. … Being in the presence of Donald McIntyre was like, one imagines, being in the presence of an intellectual nuclear reactor. Learning radiated from him. …

APL and J

Donald's publications listed near the end of the paper show 12 papers related to APL and J with all but one either being given at APL Users Meetings or published in *Vector*. These represent only a small proportion of his work on these languages given the large number of invited talks and workshops with which he was involved. In this section we shall discuss two very early papers on APL both given at APL Users Meetings in Toronto, his seminal paper published in the issue of the *IBM Systems Journal* celebrating the 25th anniversary of APL, and briefly some of his J papers.

The first of the APL papers, "The architectural elegance of crystals made clear by APL", describes the use of APL in a second-year crystallography course with a class of about 12 students. There were no prerequisites, with the necessary mathematics, including spherical trigonometry and matrix algebra, being introduced as needed. The approach to the use of APL is of interest and is best illustrated by the following quotation taken from the Introduction: "It is important always to remember that the subject is crystallography and mineralogy, not mathematics or computer programming. For this reason, I introduce notation only as needed for the work in hand, minimizing the computer and machine characteristics." Even today, twenty-five years later, this course makes a refreshing contrast to so many introductory computer courses where the examples are chosen primarily to illustrate features of the programming language being used.

The second paper, "APL in a liberal arts college", gives an account of the introduction of computers and computer courses at Pomona College. Two courses, both given by Donald, are of particular interest: one a large, far-ranging course entitled "Introduction to Computing" with about 140 students, and the other an intensive, two-week faculty seminar in computing with 12 participants. Again we see that the emphasis is on meaningful class problems which include the construction of frequency histograms, the economic problems of double declining balance and the Cobb-Douglas Production function, Brillouin's Diversity Index, an extended syllogism from Lewis Carroll, and a literary database. At the end of the paper we are reminded, in case we might become discouraged with our own attempts to introduce the APL notation or indeed any mathematical notation into our courses, that the English mathematician William Oughtred (1574-1660) faced similar difficulties when he introduced our familiar multiplication sign.

The direct form of function definition is used throughout both of these papers. Some readers may encounter old friends, possibly long forgotten, amongst these functions, only two of which will be mentioned here. The first is the recursive definition of the factorial function

```
      FAC: ω×FAC ω-1:ω=0:1
```

and, for example, `FAC 4` is 24. The second is the function

```
    POLY: (α∘.*⍳ρω)+.×ω
```

for the evaluation of polynomials, where the left argument gives the values of the independent variable and the right argument gives the values of coefficients. As an example, consider evaluating the cubic which in conventional notation would be written as $1.5 + 2x - 3x^2 + 0.5x^3$for $x = 2.5$ and 3.5. Then if `C ← 1.5 2 ¯3 0.5` and `X ← 2.5 3.5`, the expression `X POLY C` would have the value `¯4.4375 ¯6.8125`. (In J this calculation could be done using the polynomial verb `p.` and if `c=. 1.5 2 _3 0.5` and `x=. 2.5 3.5`, then `c p. x` would be `_4.4375 _6.8125`.)

Donald's paper "Language as an intellectual tool: From hieroglyphics to APL", which appeared in the special issue of the *IBM Systems Journal* (vol. 30, no. 4) celebrating the 25th anniversary of APL, is a masterly survey of the development of mathematical notation interspersed with many examples given in APL, usually in both ordinary functional form and in direct definition form, and in J. They include a sequence of definitions for the familiar statistical computations of mean, deviations from the mean, sum of squares, etc., the calculation of pi using Archimedes' method of inscribed polygons and the calculation of interest payments on a declining balance as examples of recursion, and some examples from the logical calculus of George Boole. This is followed by a detailed account of the evolution of the concept of an array in one or more dimensions with reference to the work of the nineteenth-century English mathematicians J. J. Sylvester, William Cayley, and William Rowan Hamilton. The last section, "Notation as a tool of thought" which is the title of Ken Iverson's Turing Award lecture, presents the views on the importance of good mathematical notation of Charles Babbage, J. J. Sylvester, A. N. Whitehead, Bertrand Russell and Giuseppe Peano. There is a bibliography of 125 cited references and notes followed by 92 general references. We should mention that Donald's paper is followed by Ken Iverson's "A personal view of APL" which presents his views on the language and the development of J. Ken remarks that he turned his attention to developing a dialect of APL when he "retired from paid employment", a delightful phrase that applies only too well to many academics and other professionals. Anyone interested in the development of APL and J would be advised to read these two landmark papers.

Donald's papers on J can be divided into two groups, those concerned primarily with the language and its differences with APL and those concerned with applications. In the first group is an introduction to J for persons familiar with APL and papers on special topics such as function composition and the use of conjunctions. In the second group there are papers on the use of J in teaching elementary arithmetic, eigenvalue calculations, and importing data into a J system. All of these papers show a remarkable appreciation of the language, especially the first one, “Mastering J”, written in 1991 at time when most of us were just becoming aware of the language and were reluctantly abandoning the APL symbols with which we were so comfortable.

We shall mention, and only briefly, one of the J papers, “Perils of subtraction: a new language for an old algorithm” which begins with a discussion of the calculation of pi by Archimedes’ method of inscribed polygons. It might be considered a continuation of a discussion of the same topic in the paper “Language as an intellectual tool” which was published in the same year. In this latter paper it was shown that the value after 8 doublings of the number of sides the value of pi was 3.14159 and a remark is made at the end of the section that the “algorithm fails when the number of doublings is further increased”. However, in the “Perils of subtraction” paper it is shown that with 28 doublings, giving a polygon of with 1,610,612,736 sides, calculations in J give a value of 0 for pi, a result explained by a discussion of number representation in a PC and the process of subtraction.

Perth

When he retired from Pomona College in 1989 after 35 years service, Donald, Ann and Ewen returned to Scotland and settled first at Kinfauns near Perth and then later in the centre of Perth. Interpreting retirement by Ken Iverson’s phrase quoted above he continued with his work with APL and the newly developing J, and with giving invited papers and workshops. His professional work was recognized during this time and he was made a Fellow of both the University of Edinburgh and the University of St. Andrews. He also found time to study Egyptian hieroglyphics and was able to translate a few symbols while visiting an active Coptic church in Scotland.

Donald’s last major work in computing was an account of the development of the APL and J languages which was intended when completed for submission to the *Annals of the History of Computing*. A rough draft of 29 pages, entitled “The Story of APL and J”, gives a fascinating account of Ken Iverson’s childhood in rural Alberta, his awakening interest in mathematics in school, and, after wartime service in the Royal Canadian Air Force, his undergraduate education at Queen’s University and his graduate work at Harvard leading to a Ph.D. under Howard Aiken. The evolution of the APL language is described up to the publication of *A Programming Language* and *Automatic Data Processing*. He worked closely with Ken on this paper up to a few days before Ken’s death.

Fortunately Roger Hui has edited these notes and they have been published, with an introduction by Roger, in *Vector* in Ken's name as "An autobiographical essay".

Donald was also active in the Perth community. He was Chairman of the Perth Civic Trust and helped save Dunsinane Hill – of Shakespeare's Macbeth fame – from being quarried to destruction, was an active member of the Burns Club, and was honorary archivist of the Scottish Mountaineering Club. He also took piping lessons and had the reel "Professor Donald McIntyre" written in his honour.

Epilogue

In his last years Donald suffered from Parkinson's disease and vascular dementia. He died in Perth on 21 October 2009. The memorial service, held in St John's Kirk (founded in 1248), was introduced by the playing of the reels "Professor Donald McIntyre" and "Lord Lovat's Lament" and celebrated his Scottish and Church of Scotland heritage, his professional life as a scholar, geologist and historian, and his family and community life in Perth. To the scripture readings from the Old and New Testaments read at the service we might append here the following from Ecclesiasticus, ch. 32, v. 8 which reflects the spirit, if not the letter, of the APL and J languages: "Let thy speech be short, comprehending much in few words."

Acknowledgements

Donald's beautifully crafted and carefully written pages on the World Wide Web contain a wealth of information on his life and work. The Home Page at www.mcintyre.me.uk with its many links has a large colour picture of the beautiful Perth Bridge over the River Tay which, we are told, is a "Five minutes walk from home!".

I would like to thank Ann McIntyre for her encouragement and help throughout the preparation of this paper, Roger Hui for his most helpful comments, and Willem Langenberg for providing a copy of Donald's 1987 Convocation address in mp3 format.

I first met Donald at the APL Users Meeting, "The March on Armonk", in Binghamton in 1969. Since then I have valued not only his friendship but also his guidance given by "precept and godly example", if I may borrow a phrase from our common heritage. His help to me and to my work has been out of all proportion to the very little I have been able to do for him. Thank you so very much, Donald.

Publications

1. Craig, G. Y., Donald B. McIntyre and Charles D. Waterston, 1978. *James Hutton's Theory of the Earth. The Lost Drawings.* Scottish Academic Press, Edinburgh.

2. McIntyre, Donald B., 2008. "James Hutton, the Clerks of Penicuik and the igneous origin of granite." *Transactions of the Royal Society of Edinburgh*, vol. 15, Supplement 1-Supplement 15.

3. McIntyre, Donald B., 1997. "James Hutton's Edinburgh. The historical, social and political background." *Earth Sciences History*, vol. 16, no. 2, pp. 100-157.

4. McIntyre, Donald B. and Alan McKirdy, 2001. *James Hutton. The Founder of Modern Geology.* National Museums of Scotland Publishing Limited, Edinburgh.

5. McIntyre, Donald B., 1986. "The architectural elegance of crystals made clear by APL." *APL Users Meeting Proceedings*, Toronto, Ontario, pp. 233-250.

6. McIntyre, Donald B., 1980. "APL in a liberal arts college." *APL Users Meeting Proceedings*, Toronto, Ontario, pp. 544-581.

7. McIntyre, D. B., 1991. Language as an intellectual tool: From hieroglyphics to APL." *IBM Systems Journal*, vol. 30, no., 4, pp. 554-581.

8. McIntyre, Donald B., 1991. "Mastering J." *APL Users Meeting Proceedings*, Stanford, California, pp. 264-273.

9. McIntyre, Donald B., 1992. "Hooks and forks and the teaching of elementary arithmetic."*Vector*. The Journal of the British APL Association, vol. 8, no. 3, pp. pp. 101-110.

10. McIntyre, Donald B., 1992. "Using J with external data: Two examples." *Vector*. The Journal of the British APL Association, vol. 8, no. 4, pp. 97-110.

11. McIntyre, Donald B., 1993. "Using J's boxed arrays." *Vector*. The Journal of the British APL Association, vol. 9, no. 1, pp. 92-124.

12. McIntyre, Donald B., 1993. Jacobi's method for eigenvalues: An illustration of J." *Vector*. The Journal of the British APL Association, vol. 9, no. 3, pp. 125-133.

13. McIntyre, Donald B., 1995."Perils of subtraction: A new language for an old algorithm."*Vector*. The Journal of the British APL Association, vol. 11, no. 44, pp. 93-103.

14. McIntyre, Donald B., 1995. "The role of composition in computer programming." *APL Users Meeting Proceedings*, San Antonio, Texas, pp. 116-133.

15. McIntyre, Donald B., 2001. My Involvement in the Use of Computers. 14 pp. (unpublished)

16. McIntyre, Donald B., 2005. The Story of APL and J. 29 pp. (unpublished)

17. McIntyre, Donald B., 2006. “A tribute to Ken Iverson.” *Vector*. The Journal of the British APL Association, vol. 22, no. 3, pp. 109-114.

Additional references

1. Alexander, David, 2010. “Professor Donald McIntyre.” *Pomona College Magazine*, vol. 46, Winter issue, p. 59.

2. Butcher, Norman E., 2009. “Donald McIntyre. Mountaineer, geologist, scholar and teacher.”*The Scotsman*, November 13.

3. Iverson, K. E., 1991. “A personal view of APL.” *IBM Systems Journal*, vol. 30, no. 4, pp. 582- 593.

4. Iverson, Kenneth E., 2008. “An autobiographical essay.” *Vector*. The Journal of the British APL Association, vol. 23, no. 4, pp. 70-84.

5. Merriam, Daniel E., 2010. Memorial to Donald B. McIntyre (1923-2009). *Geological Society of America Memorials*, vol. 39, pp. 23-25.

6. Phillips, Lucy Dickson, 2010. “Piping professor.” *Pomona College Magazine*, vol. 46, Winter issue, p. 59.

7. Smillie, Keith, 2011. “Donald McIntyre: Geologist, historian and array language advocate 1923-2009.” *Annals of the History of Computing*, vol. 33, no. 1, pp. 73-77.

Industry news

Dyalog Ltd

Version 13.0

Dyalog APL Windows Version 13.0 was made commercially available on April 1 in 32 and 64 bit versions. The AIX and Linux versions are also commercially available.

Again this year, Dyalog has kept the pricing unchanged, which means prices have now stayed the same year on year since 2006.

For more information on new features and functionality please visit www.dyalog.com/v_13-0.html or visit the documentation page for Version 13.0 on http://www.dyalog.com/documentation/13.0

World Wide Computer Programming Contest 2011

Now in its third year, the Programming Contest for 2011 was kicked off on May 12. The purpose of the contest is specifically to encourage students and others to investigate APL.

A fully featured copy of the latest release of Dyalog APL is made available free of charge to students, whether or not they wish to participate in the contest. Contest participants can either use the Microsoft Windows or the Linux versions of Dyalog APL, or any other version of APL as long as it is converted to the Dyalog APL platform. Tools are provided on the contest website for interchange format conversion.

A total of US$12,600 in prize money has been provided by several sponsors, including US based Fiserv, Italian based APL Italiana, Danish based SimCorp, Dyalog Ltd., as well as several anonymous individuals and companies.

The First Prize winner can look forward to US$2,500 plus a round trip travel from anywhere in the world where the winner lives to the Dyalog '11 Conference in Boston Massachusetts, USA, during October 2-5 2011, where they will receive the award in person and present the work that lead to winning the prize.

Second and third prizes will be awarded with US$1,200 and US$600 respectively and a further 20 contestants will receive US$100 each. Additionally, the people or organisations that introduce the winning students to the contest will receive the same dollar prizes – and they need not be students to make the introduction.

This year we're making more use of the social networking sites Facebook and Twitter to post information regarding the programming contest. We also regularly update the Q&A page relating to the Contest on http://www.dyalog.com/contest_2011/qanda.html

The deadline for submission is 12:00 UTC (Noon), Sunday August 14, 2011. The winners of the three main prizes will be notified directly via e-mail and their names will be posted to www.dyalog.com on August 22nd 2011.

Dyalog '11 – THE Array Language Event of the Year

Book your diaries already now for October 2-5, where the Dyalog User Conference will take place in Boston, Massachusetts, USA.

This year's conference will take place at John Hancock Hotel & Conference Center located in the heart of Boston's Back Bay. It is a small venue - in comparison to many traditional Conference venues – only 68 guest rooms, so the Dyalog '11 Conference will occupy virtually the entire venue. We also urge you to book accommodation when you register, as we cannot guarantee bedrooms after the expiry of Early Bird Registration. Should we run out of guest rooms there are fortunately several other hotel located in close walking distance to John Hancock.

We're in the process of putting the conference program together, but it will follow the now traditional format with Training Courses on Sunday 2 October and Wednesday 5 October pm. We can already now see that we will have a fully packed programme of Conference sessions – presented by invited speakers, users and Dyalog developers throughout Monday, Tuesday and Wednesday a.m. Remember to pack your glad rags for the banquet dinner Tuesday 4 Oct.

Conference participants can also look forward to a very exciting conference edition of Dyalog APL containing a number of experimental features from the Dyalog Research Labs.

The Registration System is now open and there is a 10% discount for Early Bird registration. Early Bird expires on August 14th at 23:59 UK time.

For more information on the Dyalog '11 conference programme and to register please go to http://www.dyalog.com/dyalog_11/index.html

Manpower

After having worked with us on a consulting basis over the last couple of years, we are delighted that **Brian Becker** joined Dyalog full time in February. As you may know, Brian has in particular been working on the Stand Alone Web Service (SAWS) – on which he conducted a workshop at the APL 2010 Conference in Berlin. Brian will play a key role in the new “APL tools group” that we have formed where he will take primary responsibility for the design, implementation and documentation of tools that we will be writing in APL in the years to come. Brian is based in Rochester, New York – just south of Lake Ontario. In addition to working on APL tools, Brian is also providing helpdesk support to selected US customers outside UK office hours.

Check out the hot topics and discussions on Dyalog Forums

To follow all new discussion on Dyalog join our forums at forums.dyalog.com – you do not need to be a user of Dyalog APL, nor do you need to register, to read the postings. The Forums are getting livelier all the time and have become our primary channel for formal and informal announcements including FAQs.

MEETING

A lightweight look at Minnowbrook

by Roy Sykes (roy@roysykes.com)

The Minnowbrook series of APL Implementors Conferences was revived in 2007. Although its proceedings are always confidential, Roy Sykes gives us a glimpse of what it was like to be there.

The 2010 APL Implementers Workshop was held 16-20 October at the eponymous Minnowbrook Conference Center, Syracuse University’s retreat in the Adirondack Mountains of upstate New York. Situated on the shore of Blue Mountain Lake, complete with boathouse, docks, and canoes, Minnowbrook is an immersive experience, as Border Collie Tess (brought to herd the participants) found out the hard way.

There is always plenty of spirited discussion at Minnowbrook

For four days and four nights, the conferees explained and discussed, speculated and debated the present and future of APL, parallel programming, web services, .Net, GUIs (graphical user interfaces), and the Internet. With (mostly) all attendees attending (mostly) all sessions, which lasted from 9am to 9.30pm, followed by evening seminar, one completes the experience of Minnowbrook both exhausted and exhilarated, primed to embark on the challenges of design, implementation, education, and marketing of the growing family of languages rooted in APL.

What's said at Minnowbrook stays at Minnowbrook, so this review is somewhat oblique. Our community is strangely and wonderfully both competitive and cooperative. At Minnowbrook, marketing and recruiting plans are discussed. New features, both planned and speculative, are explained and debated.

A session in progress in the classroom

Gitte Christensen presented the positioning and marketing plans for Dyalog APL, leading to an intense debate led by Morten Kromberg. Morten presented the development and technical plans for Dyalog APL, leading to an intense debate led by Gitte. Joe Blaze discussed APL2000's plans for APL+Win and VisualAPL, leading to an intense debate led by Gitte and Morten. Alan Graham and Cory Skutt each presented their somewhat black-box plans, intensely debated by everyone.

Several implementers, led by David Liebtag of IBM, discussed their various implementations and problems with PEACH (parallel each) and related operators, leading to cacaphonous parallel discussions quashed only by the immoderate moderator, Roy Sykes. These tend to benefit SIMD (single-instruction, multiple-data) types of applications. In contrast, MIMD (multiple-instruction, multiple-data) applications benefit from the types of work Ron Murray did at Microsoft, called cohort scheduling, to address server scheduling issues, and Jacob Brickman is doing in development of function (as opposed to data) arrays. Bob Smith addressed meaty issues relating to singletons and prototypes, leading both to dinner and to subsequent discussions of multisets and related functions (union, intersection, index of, etc.), new system functions, and powerful new operators.

Dinners at Minnowbrook are superb

Shannon Bailey, a founder of Native Cloud Systems, fascinated the group with her compiled APL, which forms the heart of a secure and scalable native cloud stack, which is also inherently parallel. Both she and Alan Graham emphasized the complexity and insecurity inherent in today's computing and communication environments, and addressed solutions thereto. Paul Grosvenor also addressed solutions to the GUI issues and questioned APL's role, which led to a wider discussion of object orientation and its utility in APL for anything but GUIs.

Other topics covered in more or less detail were Unicode and symbology, APL on 64-bit processors, IDEs (integrated development environments), and broadening the APL market and educating new APL programmers, the latter particularly addressed by Ray Polivka, Gitte Christensen, Joe Blaze, and Paul Grosvenor. Jon McGrew wrapped up the final session with a video presentation of the work of Catherine Lathwell (daughter of Dick), who is creating a movie documenting the history of APL, and urged the group to keep in contact with “http://AProgrammingLanguage.com” for ongoing information about the project.

We decided to hold Minnowbrook every odd year starting in 2011, which is already scheduled for 20-24 September, when the autumn colours will be on full display and the weather a bit less brisk.

Two pontoon planes were brought in for us

What little recreational and eating time remained was pleasingly filled with aeroplane rides around the Adirondacks (graciously donated by Paul Grosvenor), canoeing, hiking, excellent food from the Minnowbrook staff, tasty wines courtesy of Joe Blaze, zesty ales supplied by Jon McGrew, multiple libations supplied by Garth Foster, and scrumptious malt Scotch whiskies sacrificed to the crew by Paul Grosvenor and Roy Sykes. Jon McGrew did his usual excellent work providing the materials for the workshop. Finally, we thank Garth Foster, who started the Minnowbrook conferences in the early 1970s, and without whom they would not exist today.

A serene time: Minnowbrook at midnight (60-second time-exposure)

MEETING

FinnAPL Spring Meeting 2011 at Vanajan Linna

by Adrian Smith (adrian@apl385.com)

FinnAPL celebrates 30 years

This meeting was a good mix of history (both of FinnAPL and the various statistics associations that have always formed the backbone of the group) and some very leading-edge material, like Tomas with his boat simulator, using the 8-way parallel capabilities of the GPU to run part of the physics model.

Vanajan Linna

As ever, the location was part of the point, although much of the European Tour golf course was still under snow. However it was almost shirt-sleeve warm and the snow was subliming away as you watched, so we both came back a lot browner then we were on arrival!

First tee

I can see why the Finns are so good at creating usable devices (like Nokia phones) – they have such a simple pragmatic approach to problem-solving, always taking the shortest route to a good result. They are natural ‘extreme’ programmers, finding a small corner of a problem which has a known need and a simple solution, implementing that, and moving on carefully to the less sure parts. Maybe a lifetime of ice-fishing has something to do with this? Put one foot in the wrong place and you get very cold rather quickly!

Meeting notes (day-1)

A History of Astika
Jaakko Ranta (Statistics Finland)

Jaakko with one of the early APL2/PC implementations

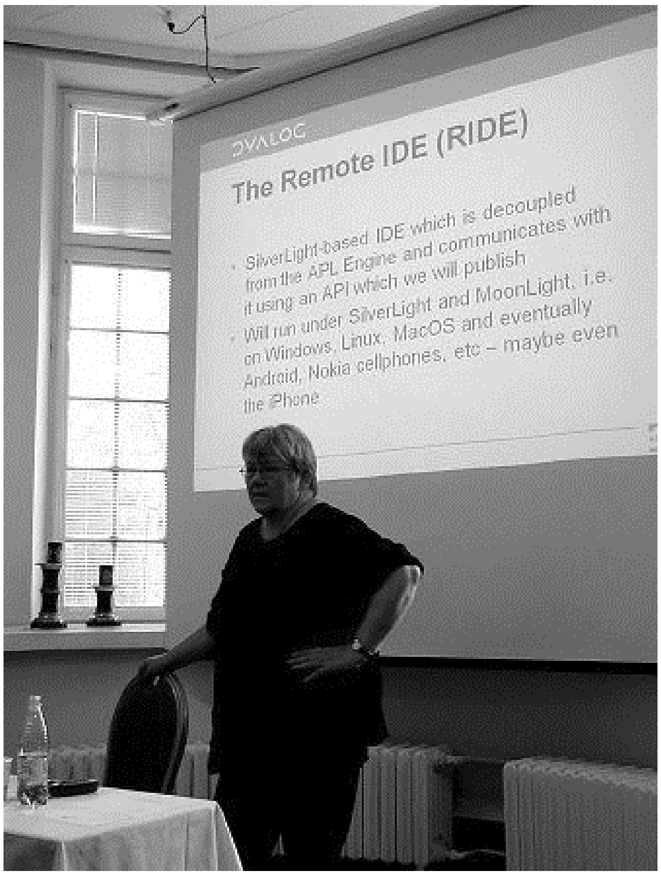
Astika is a time-series database, which began in 1979, after one year in the planning. APL was widely available (and was being actively sold by IBM) so VS APL running on VSPC was the natural choice for an interactive system. It started life on 300baud terminals, but soon moved to 3270 displays, with printed (and commonly memorised) code catalogues for cheap and efficient access. The internet edition began in 1995, by which time over 30,000 timeseries were available, requiring two big folders for the codes (but many customers just knew the codes they used!).

From 1985 it was available in English, and from 1987 real-time updates of industry data began. APL*PLUS/PC could be used to view the data, but at a cost of 1 interpreter per user it was way too expensive. An AI project was begun to enable a true ‘natural language’ user-interface; this was eventually abandoned, but it left behind a well-defined query language which may still be used to this day.

PC-Astika (1990-2010) ran in APL2/PC (free packager and runtime), in parallel to an early Windows system (written in Visual Basic), until both were superseded by Astika-W in Dyalog APL from 2000 onwards. This was always a server application, using a single database on component files, with users accessing the data over TCP/IP. The data-maintenance was gradually re-written, and graphics added with RainPro.

From 2009, all the data was moved to PX-Web to fit with other national databases, but it has been hard to get rid of all the (paying) customers for the old system! They like being able to save their queries (not available in the new system). The web interface makes good use of SharpPlot to serve custom charts beside the traditional tabular data.

News from Dyalog
Gitte Christensen (Dyalog Ltd)


DYALOG
The Remote IDE (RIDE)
SilverLight-based IDE which is decoupled from the APL Engine and communicates with it using an API which we will publish
Will run under SilverLight and MoonLight, i.e. on Windows, Linux, MacOS and eventually Android, Nokia cellphones, etc – maybe even the iPhone

Gitte introducing the new Remote Interface

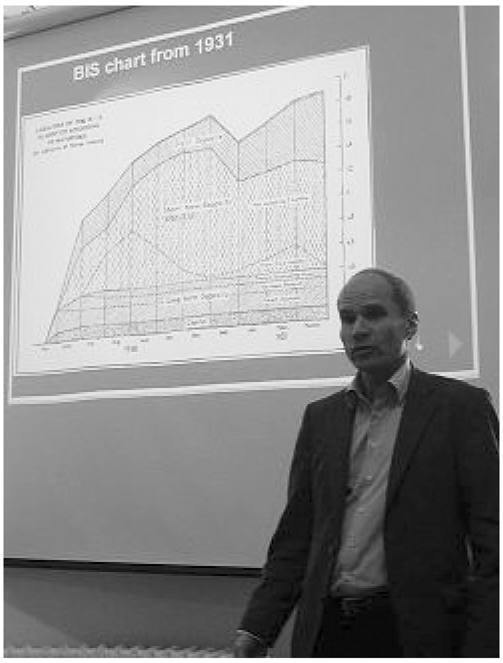
The theme of the talk was all about allowing the domain expert to work closely with (or often to be) the programmer. Eight of the ten largest customers (who account for 80% of the revenue) grew in this way, and many are now professional software vendors. Asset management is typical (Simcorp, APL Italiana, FiServ, InfoStroy) with reporting and budgeting (KCI, Carlisle Group) also prominent. Then there is ProfDoc which does healthcare for 40,000 patients in Sweden, is heading towards 1m records, and may soon be the biggest medical records system in the world. Exxon-Mobil reckon that their refinery optimiser is worth at least $50m pa (and rising with the oil price). All these are classic examples of offering an expert a tool for thinking, and waiting 10 or 15 years for a locally-developed solution to grow into something very significant.

As well as supporting established experts, the new challenge is to tease and bribe the kids with a free APL. Over 1000 have been licensed to date, with new enquiries arriving daily from students of finance, business, physics and environmental studies. The annual programming contest led to a winter internship for the winner, and maybe after his PhD he will be back! Last year's winner is helping to design this year's contest, and each one results in a run of new educational licences.

The fair pricing model – pay nothing until you make money, then a simple royalty – is working well. As customers invest more in APL, Dyalog have been able to build their full-time staff up from 5 to 20, and Gitte is confident that the company now forms a stable, viable investment for the shareholders. The next opportunity to get together will be in historic downtown Boston (John Hancock's house) from 2-5 October 2011; meanwhile more code-sharing with `]ucmd` will be encouraged, along with forum activity and even Twitter.

New applications are typically web-based, and several old customers have returned to build the next generation. Many retired APLers are taking the non-commercial licence, and are helping to enhance the interpreter by encouraging the less commercial aspects, like complex numbers and primes. Dyalog are still waiting for that blockbuster application (such as launched Python) but feel ready for it when it comes, with an integrated approach to functional, array and object-oriented programming.

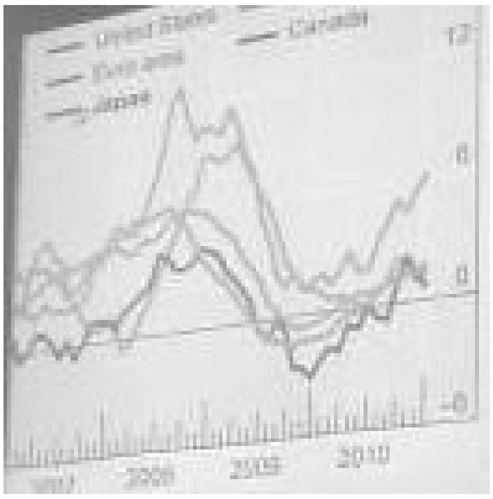
APL# and R-IDE will allow APL to run wherever a browser can run, or to be spread across machines, with the development environment on your laptop (or phone) and the interpreter on some IBM mainframe under Linux if that is what you need. Other research projects include bridging to Java, JIT compilation of Dfns, and rational numbers. Morten can fill in the details tomorrow!



Timo with a BIS chart from 1931

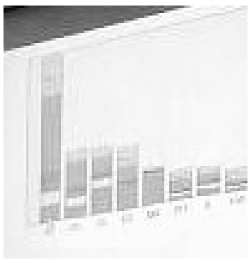
Paradigm shift – from paper to interactive
Timo Laurmaa (BIS)

Timo succinctly described his role in the BIS as "Helping the data to speak". As head of web communications and publishing, he is actively researching the paradigm shift from paper to interactive, firstly in finding out the requirements, and then thinking how to improve the tools. The toolkit must adapt to the new media, but the graphics should continue to convey the message about the data; all this must happen without a big change to the existing paper workflow.



Helping the data to speak

This shows a simple modification to one of the many line-charts from the latest BIS Quarterly Report (download the PDF from their website to see the originals). What Timo has done is greyed all the lines, and then coloured the one that the user points to with the mouse. This can easily be added to the original workflow without changing the printed image, but is a real enhancement to the SVG version.


Stacked barchart

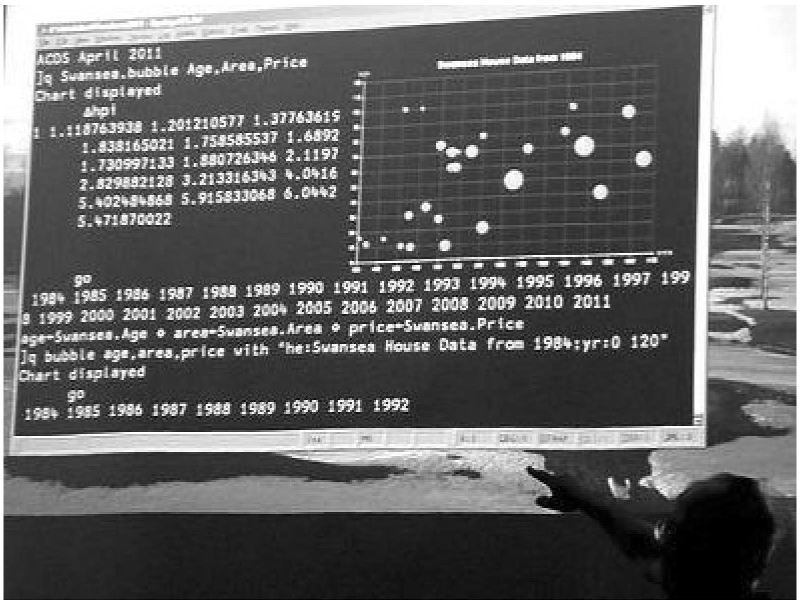
Stacked barcharts are quite bad even on paper, so applying the same simple approach to user-selection may not be enough to make this one work well. Highlighting sections in the middle of a deep stack still leaves the data hard to see, so maybe the stack could re-arrange, or the chosen series could be run out as an overlay plot just below the composite? More experiments are clearly needed here!

The other challenge is to pick up the *Wow!* element in the data and have it jump off the screen as the presenter talks in front of it. Timo showed a splendid modification to the UK interest chart which ran up to 2008 on a y-axis ranged from 3.5% to 7% – running this forward with the y-axis diving to zero to follow the plummeting data line really hit you in the face with the message “Something unheard-of just happened” which is the whole point of the presentation!

One final challenge is to find a good way of pulling out extra dimensions as the user navigates the chart. Timo’s final example was a gantt chart of the Finnair fleet showing the planes down the left, and coloured bars for the inbound and outbound legs. By pointing to a leg with the mouse, you could show the routing (useful) and show all the other legs from all the other planes which were flying the same route (simply by re-colouring the bars).

This was a good point for Timo to hand over to Adrian, who spent Wednesday working with him and Morten in an attempt to understand how SharpPlot/Rainpro might begin to implement some of these ideas.

Animations in SharpPlot
Adrian Smith (Causeway)



```
ACDS April 2011
]q Swansea.bubble Age,Area,Price
Chart displayed
      ahpi
1 1.118763938 1.201210577 1.37763615
      1.838165021 1.758585537 1.6892
      1.730997133 1.880726346 2.1197
      2.829882128 3.213316343 4.0416
      5.402484868 5.915833068 6.0442
      5.471870022

      go
 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 199
8 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011
age←Swansea.Age ⋄ area←Swansea.Area ⋄ price←Swansea.Price
]q bubble age,area,price with "he:Swansea House Data from 1984:yr:0 120"
Chart displayed
      go
 1984 1985 1986 1987 1988 1989 1990 1991 1992
```

Adrian with the Swansea house-prices re-cast as exploding bubbles

This talk was largely born out of a full day spent chatting with Timo and Morten the day before the conference began. We started by exploring the requirements for animations, and then developed a couple of simple demos to show what current technology could achieve easily (well, in less than 6 hours after a few beers!).

Incidentally, the APL session in the picture (taken by Timo) nicely illustrates the use of `]ucmd` to allow APL to behave as a simple SQL engine for data exploration. The SQL syntax (at least the variant of it devised by Arthur Whitney called q) is ideally suited to the task of drawing charts from data in the workspace, and the great thing about user commands is that the interpreter never sees your (very non-conforming) syntax, and the session syntax-colouring also ignores it.

The requirements clearly begin with simple enhancements to existing paper charts. If you grab almost any PDF from www.bis.org you will see lots of classic small multiples, where a grid of similar charts (generally with identical axis ranging and a common legend) are used to compare countries or time periods. This works well on paper (really good colour discrimination, massive resolution) but really badly on a typical laptop display, and is utterly useless on a phone. By creating these as a stack of pages, with a simple user-driven scroller or page control, the problem is largely solved – minimal change to the paper workflow, a simple mod to SharpPlot to output the trellis as a paged stack, and this should be achievable in almost any medium.

A variation on this theme is where there are lots of lines on a timeseries, or a big stack of bars on a barchart. On paper, colour discrimation seems to work much better, and the user can generally follow a line without difficulty. On screen you need to add some (simple) JavaScript so the user can pick out the line of interest, having drawn them all in a low-contrast grey to begin with. This is very easy in SVG or XAML, but hard to do well if you are limited to raster images (PNG or JPEG) where some experimentation will be needed to see how seamless this can be made to feel. Again, it should be possible without a complete re-design of the original chart – Timo is still aiming to leave the BIS workflow unaltered as far as possible.

The prime use of *animation* seems to be for presenter emphasis, rather than as a passive or user-driven display on the internet. There may be good examples out there where (for example) animation of the time dimension adds visibility to the data, but all the 'cool stuff' on Google Docs seems to leave the end-user in control of the hidden dimension. On the other hand, when a presenter is standing in front of a moving chart, which is well-timed to add punch to the point being made, the effect is very powerful and adds genuine value. A simple barchart can be timed to grow in the time it takes for the presenter to tee-up a rhetorical question, and then deliver the punch-line for him. This can be done now with SharpPlot and SVG animation, but (as Adrian illustrated by messing up several times) it requires a good knowledge of the SMIL tags and a strictly logical mind to get all the parts to work well together. Here is the finished example in SVG:

http://www.apl385.com/finnapl2011/growbar.svg

Have a look at 'View, Page source' to see all the `animate` tags which the programmer either added in SharpPlot/RainPro as *Effects*, or post-processed with a text editor in the generated SVG.

There may well be a use for simple animation for smoothing out the transitions when a chart steps through time, and the steps might involve uncomfortably large jumps between frames. The photo at the top of this report shows a faked growth pattern in Alan Sykes' data from ASLGreg, first shown at Swansea in 1984. All I did was create a bubble-chart of the prices, then hammer around a tight loop applying the Nationwide house-price index to the numbers.

```
⍝ Base data
age←Swansea.Age ⋄ area←Swansea.Area ⋄ price←Swansea.Price
:For yr :In ⍳⍴Δhpi
   age←Swansea.Age+yr
   price←Swansea.Price×Δhpi[yr]
   ⍞←' ',⍕1983+yr
   ⎕DL 1
:End
```

Crude, but has quite a good element of suspense, and a suitable 'Wow factor' as the market suddenly explodes around 2004. This could implement very well in SVG (with animations to smooth the movement and growth of the bubbles from frame to frame) or maybe just as a stack of images with some trivial JavaScript to work down the stack. Switching the visibility of each image should give a relatively flicker-free transition on this kind of progressive time-series.

Summary – lots of possibilities, and the technology is catching up fast (all the top browsers now do SVG, and everything with the SilverLight plugin does XAML). Ideas welcome, prototyping starts here.

The StormWind simulator
Tomas Gustafsson (Stormwind Ab Oy)

Tomas navigates safely home

Yes, we have seen this before, but what we was today is a massive step forward in realism and visual quality. Partly it is all about hardware – graphics cards are where much of the best research is going these days, and system memory is no longer a constraint on design. The version we saw was developed with sponsorship from the Finnish maritime organisation, and is a free download if you want to test your boat-driving skills. It has full digital charts for Finnish coastal waters, so there are plenty of islands to run into.

Essentially, it gives a step-by-step introduction to what happens in your smart new boat, from the list of mandatory safety equipment to the recognition of multi-sector lights and buoys by night. The data-processing challenge was to create virtual reality out of a truly obscure 1980s' standard for digital mapping (has anyone heard of *octets* these days?). Then there is the annoying issue wth DirectX which uses *float* rather than *double* so you only have 32 bits of precision. If your boat is 10km away from base, then a 5cm wave height cannot be expressed accurately, and things start to get lumpy. Essentially, addition doesn't work any more! So you have to shift the world on the medium scale and the boat on the small scale – it hardly matters if you get an island 5cm off!

Parallel GPU

A very neat trick Tomas used for processing speed was to implement the physics model of wave-height on the boat's motion 24-way parallel in the GPU. A little debugging display in the top left showed the sum of the forces at 8×3 points on the hull as colours (GPUs always return colours) which could easily be decoded to give the hull an appropriate kick. Certainly the realistic waves and (very) realistic cabin motion were highly impressive. And yes (before someone tries it) – if you drive flat out at an island you can beach yourself among the trees!

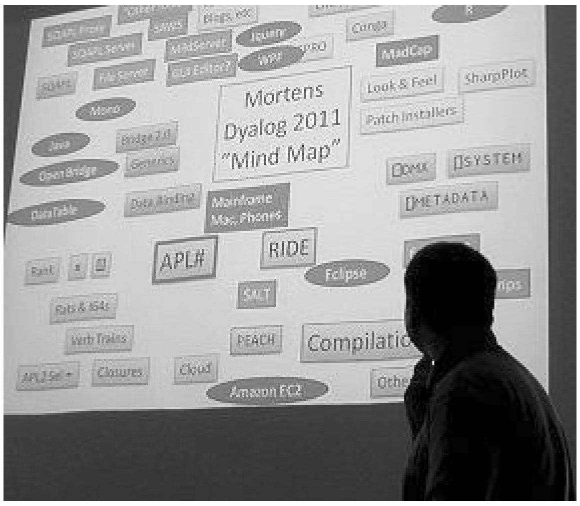
Meeting notes (day-2)

Calling Demetra+ for seasonal adjustment
Jouko Kangasniemi (EK)

Taking the seasonal effects out of quarterly data is one of those messy challenges that does not implement well in APL. A simple moving average makes a mess of the ends, and modern techniques like X12 and Arima are highly iterative. Jouko has been exploring a new library (set up as an EU project) called Demetra+ which brings together many Java and C# libraries with a common interface. It accepts XML data (easy to format with the `⎕xml` system function), or you can call the .Net interface. Using the Dyalog *Metadata* option in the Workspace Explorer is most helpful when browsing around the library. It is quite huge, with a full set of numerical methods, but far from fully documented, so expect quite a lot of status messages and `DOMAIN ERRORS` as you get the calls to work from APL.

It is redistributable, and should be much easier to set up than the R library (and its COM interface) if you need to include some of the routines as part of a shipped application.

What's cooking at Dyalog
Morten Kromberg (Dyalog Ltd)



Morten obscures a small corner of Dyalog's future

Version 13 may look much the same on the outside, but it has 128-bit float so it can do currency calculations to the nearest cent and not lose anything in the rounding. In earlier APLs:

```
      1-+/10⍴0.1
1.11022302463E¯16
```

But with version 13 you can set `⎕fr←1287 ⋄ ⎕pp←34` and zero is what you get, to 34 dec places. Of course it runs slower (only the IBM RISC machines do this in hardware) but you save so much 'check it and fix the result' code that the application may speed up, and it gets so much simpler to maintain.

With complex numbers `¯1*0.5` works as engineers always hoped it would, and the absolute limit on array size (2 billion elements) has been lifted. This is hard to test – Dyalog are upgrading a machine to 64Gb of RAM to give it a work-out! Perl-style regexp is built in now, for search and replace on text vectors, and application profiling has been made much easier to use. Short left arguments to `↑↓⌷` neaten up your code a little by filling in the spare dimensions, and you can use `⊢⊣` to chain expressions in a much more functional style than by using diamonds. The IME can help with overstrikes (by showing a drop-list) and can also be set to ignore the next keystroke, allowing Ctrl+S to make an upstile in APL and save your work in Notepad.

The mind map has nearly 50 topics on it for future work, but there are now nearly 20 people on the team, so that is only around 3 items each! This is roughly the size of group that IPSA and IBM ran with back in the days when they majored in APL, so it is definitely manageable and has been very productive in the past. Themes are speed, portability, power & simplicity, security & encryption. There will be a strong focus on tools and training materials.

Some old assumptions will have to go – in APL# parallel operators like `f¨` do not guarantee the order of execution; partial compilation will be required to make larger chunks that can run in parallel. This will work much better with Dfns, so try and aim new code towards a pure functional style if you can. APL# will be pure managed code, so it will run in the browser, or anywhere else you can use JavaScript. R-IDE will run wherever SilverLight runs (very likely on the ARM chip with Windows-8 – look out for Nokia) and the bridge idea will be extended out from .Net to Mono and Java. If you want to see more, better be in Boston in early October!

PX-Edit retrospection
Veli-Matti Jantunen (Statistics Finland)

You will have to wait for the full report on this one – I enjoyed the cartoon slides rather too much and my note-taking fell away almost to zero. A couple of technical points that I did pick up: by all means upgrade on the fly (the .exe is just a loader, pulling everything off file as `⎕OR`, but make sure to fire this off a timer, as many users leave the application running continuously). Also it is best to turn off ClearType for printing, as it can make the text look fuzzy at small sizes. Ask Veli-Matti if you need to do this – the switch is hidden deep in Windows and is not trivial to get at!

The PLURAL programming language
Walter Fil

I'm sure Walter will write this up for us when it has progressed a little further. It is an attempt to make a consistent array language out of a Java-like structure, but without all the depressing coding needed to handle simple array operations. Maybe a sort of numerical Python. I will follow its progress with interest, as he gets parts of it running to the point where I can fool around with it. It is still mostly in its PowerPoint phase, so is hard to evaluate in any useful way.

Round-up

The inevitable group photo (camera propped in the snow, hence the wide grins on many faces!)

The organisation was efficient and everything went to plan; the sun shone, the birds were singing and we saw a Camberwell Beauty in the woods near the airport, so a very successful trip. I think some good work was done, and everyone came away feeling more positive about APL than they had done for some time. So well done the Finns, see you next year!

MEETING

BAA Annual General Meeting 2011

by Peter Merritt

Minutes of the British APL Association AGM 2010 held at The Albion, 3 New Bridge Street, London EC4 on Friday 21 May 2010

Minutes of AGM 2010

The minutes had been published on the web site and were taken as read without correction.

14:40 Welcome (Phil Last)

- Apologies: Stephen (still in Japan), Paul (still in Sussex)

-16 bods in attendance (incl. guests)

14:45 Presentation 1: Morten on new features in Dyalog13

-true 128 bit numbers

-regular expressions

-large arrays

-PROFILE (successor to quad-MONITOR)

-TCPIP toolkit

-SQAPL revamp

-parallelism

15:18 AGM

-In the absence of Paul Grosvenor, Peter Merritt railroaded the meeting:-

-previous minutes (not as such; agreed anyway)

-report from chair (no)

-report from treasurer (funds OK, no sign of BCS letting go of any BAA cash)

-report from treasurer on membership

corporate was 439 members, now 415;

individual was 255, now 247 (66 UK, 180 o/s)

-thanks to Nick for splendid job

-committee elections:

In a hard-fought election lasting several seconds, exactly the same people were co-erced into doing exactly what they did last year.

Chairman	Paul Grosvenor
Treasurer	Nicholas Small
Secretary	Peter Merritt
Editor	Stephen Taylor
Activities	Phil Last
Auditor	Chris Hogan

Two others (Kai Jaeger, John Jacob) were also 'elected', until someone pointed out that those posts did not actually exist, and that to create them would require a constitutional change. They were then immediately un-elected.

AOB

- The said Kai Jaeger and John Jacob were co-opted onto the committee to continue previous functions.

- Achievement award should have been awarded... but no committee discussion had taken place. Kai therefore nominated Stephen Taylor for splendid work with Vector, seconded by loads of people; Peter Merritt to inform Paul Grosvenor to arrange 'gong'.

- Jane Sullivan proposed a vote of thanks to committee 2010 which staggeringly was accepted

15:37 AGM CLOSED

15:48 Presentation 2: John Scholes - Function Trains i.e. no more syntax errors - they're features (I've been saying that for years)

16:30 Presentation3: Mike Hughes - Application Design with WPF and APL

Mike circulated an early draft of a Dyalog manual covering these aspects.

BAA accounts and membership

Nicholas Small

Income and expenditure/receipts and payments:

	2010/11 (R&P) £	2009/10 (R&P) £	2008/09 (R&P) £	2007/08 (R&P) £
Income/Receipts				
Subscriptions	3977	1280	10711	6005
Vector advertising	0	0	250	0
Other	10	302	8142	0
	---------	---------	---------	---------
Total receipts	3987	1582	19103	6005
Expenditure/Payments				
Meetings	0	282	0	0
Administration	258	321	968	261
Vector production and despatch	4180	3725	7177	3650
Projects	201	153	169	745
Other	3	127	0	312
	---------	---------	---------	---------
Total payments	4643	4608	8314	4968
Assets summary:				
Bank and other balances	7410	7882	10789	26773
Debtors	1000	3207	5107	4247
Creditors	-3015	-7958	-10879	-7166
	---------	---------	---------	---------
Net assets	5395	3131	5017	23855
Written off			100~	500~

~ Cancelled invoice

BAPLA membership at May 2011 (previous year's figures in parentheses)

	UK				FOREIGN				TOTAL			
	Number		Vectors		Number		Vectors		Number		Vectors	
Sustaining*	6	(6)	28	(28)	5	(5)	42	(42)	11	(11)	70	(70)
Corporate*	2	(3)	10	(15)	2	(2)	15	(15)	4	(5)	25	(30)
Corp. Ind*	5	(5)	5	(5)	2	(8)	2	(8)	7	(13)	7	(13)
Individual	66	(70)	65	(70)	181	(185)	182	(186)	247	(255)	247	(256)
Non-voting	15	(19)	15	(19)	0	(0)	0	(0)	15	(19)	15	(19)
Life	1	(1)	1	(1)	0	(0)	0	(0)	1	(1)	1	(1)
Library	0	(0)	0	(0)	5	(5)	5	(5)	5	(5)	5	(5)
Russians					11	(11)	11	(11)	11	(11)	11	(11)
APL Groups					12	(12)	34	(34)	12	(12)	34	(34)
											415	(439)

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

BAA AGM 2011 – Chairman’s report

by Paul Grosvenor - June 2011

Another year passes and it seems even faster than the previous year if that is possible. Once again we held our AGM at the Albion in London to an audience of 20 or so. Unfortunately I was unable to attend at the last minute so our new Secretary, Peter Merritt, stood in for me – trial by fire and all that. After the AGM presentations by Mike Hughes (Using WPF with Dyalog APL), Morten Kromberg (New features of version 13) and John Scholes (Function-trains in APL) were all well received as was the beer in the bar afterward.

2010/2011 has, once again, been a very busy year for the APL world. The Vendors are keeping our world alive with new features and initiatives; I just wish I could keep up with all of them myself. The positive momentum I described last year still seems to be in existence and even in these troubled times new APL users continue to appear. However we must not be complacent and the good old APL flag needs to be kept waving; a subject that I know is close to many hearts.

I thought that we might be able to have a conference this year as in 2009 but pressures of work simply took over and it became impossible to find enough time to organise it properly. Hopefully an event in London in 2012 will be possible.

Our committee for 2011/2012 remains unchanged from 2010/2011 so thank you to all for your continued support. As always we welcome new blood into the committee so if you do have some spare time and want to help please get in touch. Contact details can be found at the back of *Vector*.

Our issue with the BCS regarding our funds continues but their stance has not changed. Circa £14,000 remains under their control and we will continue to apply pressure on them to release it to us. Some of you may be aware that there was a vote of no-confidence in the BCS management last year and an EGM was called as a result. I submitted some information to that meeting on behalf of the

BAA. Rather than me repeating all that was said and done please follow the following link where all the information can be found;

www.bcsegm.blogspot.com

We will keep you up to date with progress.

The BAA London group have continued their meetings each month which is very good news and I hope that even more of you can try to attend. Phil Last will be announcing each meeting through various forums including comp.lang.apl so if you are in the area, please feel free to drop in on them.

For the second year running I am proud to be able to announce that we have decided to present our **'Outstanding Achievement Award'** once again. The award is designed to acknowledge the efforts of an individual, or organisation, within the world of APL where a particularly high level of achievement has been made. This year I am very pleased to be able to award it to Stephen Taylor for his work promoting APL and in particular for his efforts building and maintaining the Vector web site and of course the hours of time he has spent editing the Vector publication over the years.

Congratulations and Thank You Stephen

Thanks also to Optima Systems Ltd for sponsoring this award. The award reads;

To

Stephen Taylor

The Outstanding Achievement Award

May 2011

From
BAA

I hope that you, like myself, are looking forward to the coming year with eager anticipation and that I may get to see even more of you in some of the forthcoming APL conferences.

As always if you have any comments, thoughts or suggestions please email me at:

chairman@vector.org.uk

D I S C O V E R

Unicode and related subjects in APL2

by Kyosuke Saigusa (JCE01163@nifty.com)

Recent service levels of IBM Workstation APL2 V2 for Windows introduced support for Japanese characters in Unicode. APL tools are described for exploiting this capacity and for introducing Japanese programmers to APL.

The APL we use

Unicode seems to have been included in the design of this product from the beginning, but before its service level 7, released in 2005, we could neither key in nor display Japanese characters in APL functions. Therefore we developed our own system based on SHIFT-JIS (ASCII) with a customised Japanese font called *APL2KJ* and a special function editor. This font was created by sacrificing some infrequently used characters.

During 2005-2007, I closely watched IBM’s efforts to implement Japanese language support with Unicode, initially with suspicion, but later with admiration, as they released Service Levels 7, 8, 9, 10 and 11, with vital issues, such as distinction between legal and illegal blanks, getting solved step by step.

As a result, two items remained unsolved because of complexity of fixing the interpreter code. One was the entry and display of Japanese characters on the APL session manager screen. The other was Japanese texts in the format masks for primitive function “format (⍕)”.

The first item above may be something that classical APL users, using APL in desk calculator mode mainly, may want, but today’s APL systems offer highly sophisticated editors to write APL programs and the dependency on the session manager is relatively smaller than before. For the data input and display in APL applications, dialogue windows will provide much better human interface.

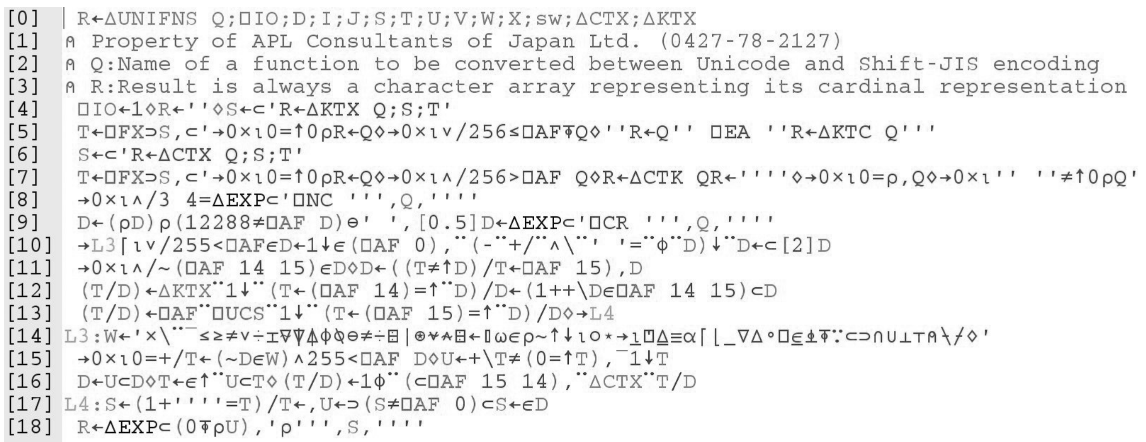
The second item is something that programmers can get around easily, by defining a function to achieve it. Therefore we judged that we can do without them.

Microsoft Windows offers a Unicode font called *Arial Unicode MS*. This font we found is not suitable to write APL functions with, because of the unfamiliar shapes and unbalanced sizes of its APL characters and pitches. New system font *Courier APL2 Unicode* on the other hand behaved as if it contained Japanese characters as well under Windows XP and Vista, and we found it a perfect font for our use.

Conversion of APL functions from ASCII encoding to Unicode

When I confirmed that the Unicode approach is the right direction, at least for Japanese programmers, I decided to convert all of my functions to Unicode encoding without exception, though IBM Workstation APL2 allows coexistence of both encodings in the same workspace, mainly for simplicity. Workstation APL2 V2 seems to handle this conversion automatically, but we had to process Japanese texts preceded and followed by SO/SI codes in APL functions in our Japanese text support system.

The function shown below was used to convert individual functions one by one with care. It produces only Unicode character arrays as cardinal representations, because if conversion fails, we can analyse to see which parts to mend. Fortunately it worked well. In actuality, it was used in the way that all the functions be converted to vector of cardinal representation arrays and they were in turn ⎕FX¨ed to produce the vector of fixed names or numeric values indicating the fix failed for the corresponding functions.

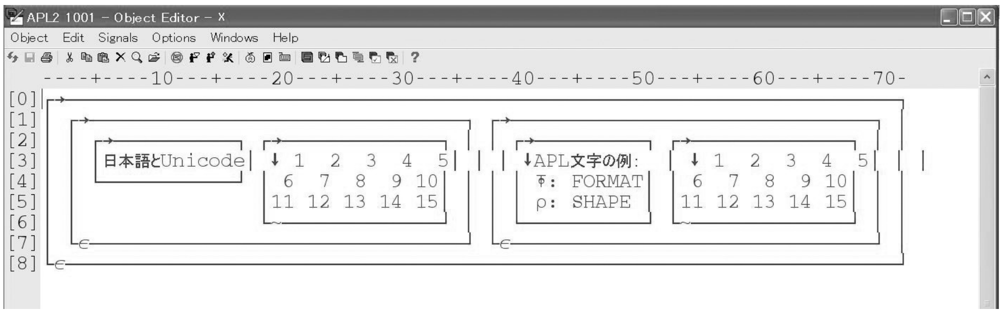


```
 R←∆UNIFNS Q;⎕IO;D;I;J;S;T;U;V;W;X;sw;∆CTX;∆KTX
⍝ Property of APL Consultants of Japan Ltd. (0427-78-2127)
⍝ Q:Name of a function to be converted between Unicode and Shift-JIS encoding
⍝ R:Result is always a character array representing its cardinal representation
 ⎕IO←1⋄R←''⋄S←⊂'R←∆KTX Q;S;T'
 T←⎕FX⊃S,⊂'→0×⍳0=↑0⍴R←Q⋄→0×⍳∨/256≤⎕AF⍕Q⋄''R←Q'' ⎕EA ''R←∆KTC Q'''
 S←⊂'R←∆CTX Q;S;T'
 T←⎕FX⊃S,⊂'→0×⍳0=↑0⍴R←Q⋄→0×⍳∧/256>⎕AF Q⋄R←∆CTK QR←''''⋄→0×⍳0=⍴,Q⋄→0×⍳'' ''≠↑0⍴Q'
 →0×⍳∧/3 4=∆EXP⊂'⎕NC ''',Q,''''
 D←(⍴D)⍴(12288≠⎕AF D)⊖' ',[0.5]D←∆EXP⊂'⎕CR ''',Q,''''
 →L3⌈⍳∨/255<⎕AF∊D←1↓∊(⎕AF 0),¨(-¨+/¨∧\¨' '=¨⌽¨D)↓¨D←⊂[2]D
 →0×⍳∧/~(⎕AF 14 15)∊D⋄D←((T≠↑D)/T←⎕AF 15),D
 (T/D)←∆KTX¨1↓¨(T←(⎕AF 14)=↑¨D)/D←(1++\D∊⎕AF 14 15)⊂D
 (T/D)←⎕AF¨⎕UCS¨1↓¨(T←(⎕AF 15)=↑¨D)/D⋄→L4
L3:W←'×\¨¯≤≥≠∨÷⌶⍒⍒⍋⌽⍉⊖≠÷⌹|⍟⍱⍲⌹←⌷⍵∊⍴~↑↓⍳○⋆→⍸⎕⍙≡⍺⌈⌊_∇∆∘⎕⋸⍎⍕∵⊂⊃∩∪⊥⊤⍝⍀⌿⋄'
 →0×⍳0=+/T←(~D∊W)∧255<⎕AF D⋄U←+\T≠(0=↑T),¯1↓T
 D←U⊂D⋄T←∊↑¨U⊂T⋄(T/D)←1⌽¨(⊂⎕AF 15 14),¨∆CTX¨T/D
L4:S←(1+''''=T)/T←,U←⊃(S≠⎕AF 0)⊂S←∊D
 R←∆EXP⊂(0⍕⍴U),'⍴''',S,''''
```

Fig.1: ∆UNIFNS source

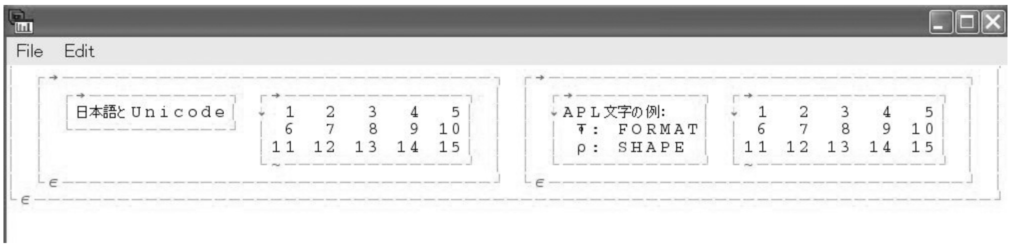
Once they are converted to Unicode encoding, APL2 system’s object editor handled the entry and display of Japanese characters in a very natural way. One of the things I found it superior to my old system was that now I didn’t have to pay attention to break Japanese characters (formerly two bytes each) into illegible segments in APL statements.

How we handle block letters for display and print


APL2 1001 – Object Editor – X
Object Edit Signals Options Windows Help

```
 ----+----10---+----20---+----30---+----40---+----50---+----60---+----70-
┌→──────────────────────────────────────────────────────────────────────┐
│ ┌→──────────────────────────────────────┐ ┌→─────────────────────────┐ │
│ │ ┌→────────────┐ ┌→──────────────┐   │ │ ┌→──────────┐ ┌→──────────────┐ │ │
│ │ │日本語とUnicode│ ↓ 1  2  3  4  5│   │ │ │↓APL文字の例:│ ↓ 1  2  3  4  5│ │ │
│ │ └─────────────┘ │ 6  7  8  9 10│   │ │ │ ⍕: FORMAT│ │ 6  7  8  9 10│ │ │
│ │                 │11 12 13 14 15│   │ │ │ ⍴: SHAPE │ │11 12 13 14 15│ │ │
│ │                 └~─────────────┘   │ │ └──────────┘ └~─────────────┘ │ │
│ └∊──────────────────────────────────────┘ └∊─────────────────────────┘ │
└∊──────────────────────────────────────────────────────────────────────┘
```

Fig.2: Example of DISPLAY output as shown on the object editor of the Workstation APL2 v2

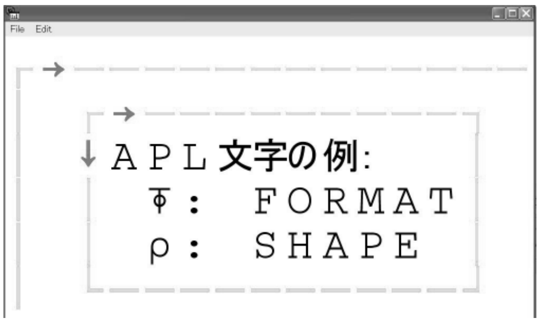


```
┌→──────────────────────────────────────────────────────────────┐
│ ┌→─────────────────────────────┐ ┌→───────────────────────────┐ │
│ │ ┌→────────────┐ ┌→──────────────┐ │ │ ┌→──────────┐ ┌→──────────────┐ │ │
│ │ │日本語とUnicode│ ↓ 1  2  3  4  5 │ │ │ ↓APL文字の例:│ ↓ 1  2  3  4  5 │ │ │
│ │ └─────────────┘ │ 6  7  8  9 10 │ │ │ │ ⍕: FORMAT│ │ 6  7  8  9 10 │ │ │
│ │                 │11 12 13 14 15 │ │ │ │ ⍴: SHAPE │ │11 12 13 14 15 │ │ │
│ │                 └~──────────────┘ │ │ └──────────┘ └~──────────────┘ │ │
│ └∊─────────────────────────────┘ └∊───────────────────────────┘ │
└∊──────────────────────────────────────────────────────────────┘
```

Fig.3: Example of ∆DISPLAY output

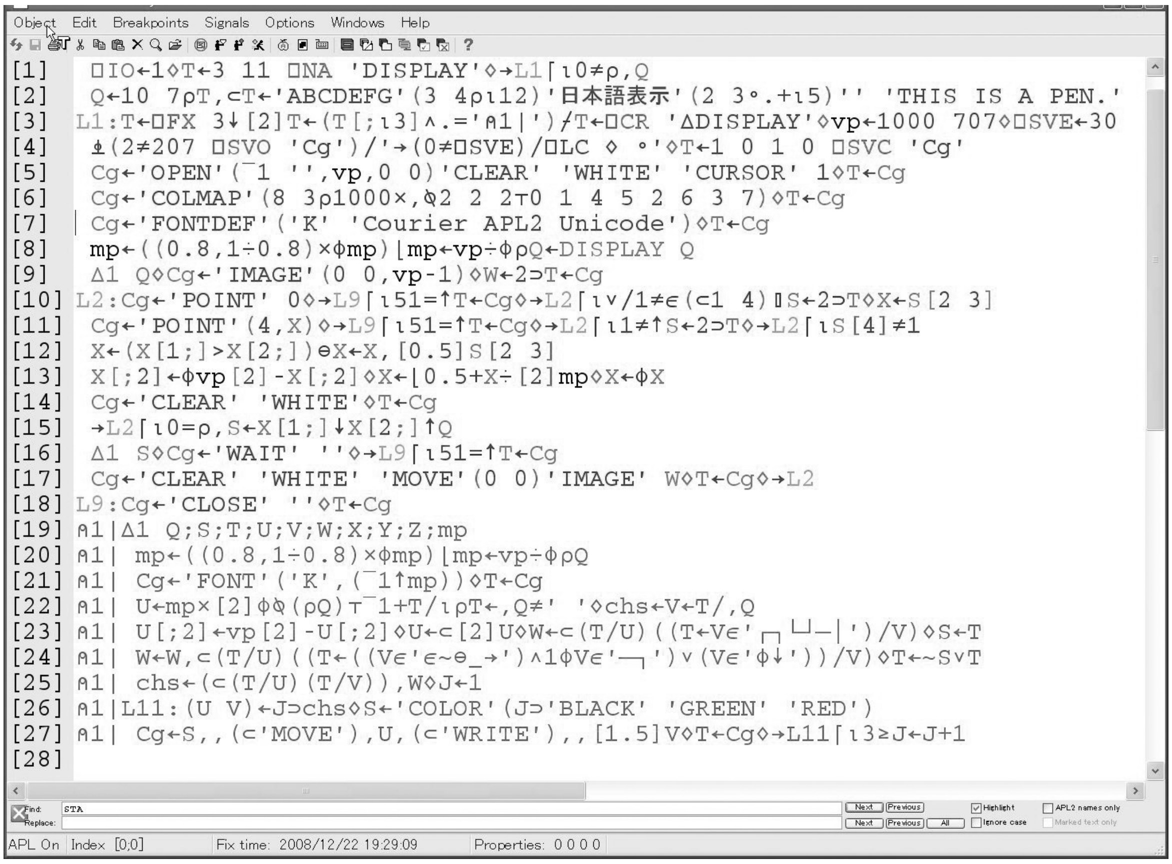
Advantage of ∆DISPLAY over DISPLAY is that the output is not affected by the size difference of the characters used. Besides it will choose the most appropriate font sizes automatically, so that it will accommodate almost any size output and any part of it can be cut to be shown in enlarged scale. This is a convenient and powerful alternative to analyse the structures of APL2 objects.

∆DISPLAY is a utility tool function programmed in APL as shown below and is stored in a name space and used from there. It can be modified easily to fit users' requirements if necessary. Internally it uses the function DISPLAY and converts its output into a graphic representation as shown in the illustration. The same logic can be applied to any similar cases to convert texts to graphics, which is more flexible.



```
┌→──────────────────
│
│   ┌→──────────────┐
│  ↓APL文字の例:     │
│   │ ⍕:  FORMAT    │
│   │ ⍴:  SHAPE     │
│   └───────────────┘
```

Fig.4: ∆DISPLAY output



```
 ⎕IO←1⋄T←3 11 ⎕NA 'DISPLAY'⋄→L1⌈⍳0≠⍴,Q
 Q←10 7⍴T,⊂T←'ABCDEFG'(3 4⍴⍳12)'日本語表示'(2 3∘.+⍳5)'' 'THIS IS A PEN.'
L1:T←⎕FX 3↓[2]T←(T[;⍳3]∧.='⍝1|')⌿T←⎕CR 'ΔDISPLAY'⋄vp←1000 707⋄⎕SVE←30
 ⍎(2≠207 ⎕SVO 'Cg')/'→(0≠⎕SVE)/⎕LC ⋄ ∘'⋄T←1 0 1 0 ⎕SVC 'Cg'
 Cg←'OPEN'(¯1 '',vp,0 0)'CLEAR' 'WHITE' 'CURSOR' 1⋄T←Cg
 Cg←'COLMAP'(8 3⍴1000×,⍉2 2 2⊤0 1 4 5 2 6 3 7)⋄T←Cg
 Cg←'FONTDEF'('K' 'Courier APL2 Unicode')⋄T←Cg
 mp←((0.8,1÷0.8)×⌽mp)⌊mp←vp÷⌽⍴Q←DISPLAY Q
 Δ1 Q⋄Cg←'IMAGE'(0 0,vp-1)⋄W←2⊃T←Cg
L2:Cg←'POINT' 0⋄→L9⌈⍳51=↑T←Cg⋄→L2⌈⍳∨/1≠∊(⊂1 4)⌷S←2⊃T⋄X←S[2 3]
 Cg←'POINT'(4,X)⋄→L9⌈⍳51=↑T←Cg⋄→L2⌈⍳1≠↑S←2⊃T⋄→L2⌈⍳S[4]≠1
 X←(X[1;]>X[2;])⊖X←X,[0.5]S[2 3]
 X[;2]←⌽vp[2]-X[;2]⋄X←⌊0.5+X÷[2]mp⋄X←⌽X
 Cg←'CLEAR' 'WHITE'⋄T←Cg
 →L2⌈⍳0=⍴,S←X[1;]↓X[2;]↑Q
 Δ1 S⋄Cg←'WAIT' ''⋄→L9⌈⍳51=↑T←Cg
 Cg←'CLEAR' 'WHITE' 'MOVE'(0 0)'IMAGE' W⋄T←Cg⋄→L2
L9:Cg←'CLOSE' ''⋄T←Cg
⍝1|Δ1 Q;S;T;U;V;W;X;Y;Z;mp
⍝1| mp←((0.8,1÷0.8)×⌽mp)⌊mp←vp÷⌽⍴Q
⍝1| Cg←'FONT'('K',(¯1↑mp))⋄T←Cg
⍝1| U←mp×[2]⌽⍉(⍴Q)⊤¯1+T/⍳⍴T←,Q≠' '⋄chs←V←T/,Q
⍝1| U[;2]←vp[2]-U[;2]⋄U←⊂[2]U⋄W←⊂(T/U)((T←V∊'┌└─|')/V)⋄S←T
⍝1| W←W,⊂(T/U)((T←((V∊'∊~⊖_→')∧1⌽V∊'─┐')∨(V∊'⌽↓'))/V)⋄T←~S∨T
⍝1| chs←(⊂(T/U)(T/V)),W⋄J←1
⍝1|L11:(U V)←J⊃chs⋄S←'COLOR'(J⊃'BLACK' 'GREEN' 'RED')
⍝1| Cg←S,,(⊂'MOVE'),U,(⊂'WRITE'),,[1.5]V⋄T←Cg⋄→L11⌈⍳3≥J←J+1
```

Find: STA
Replace:

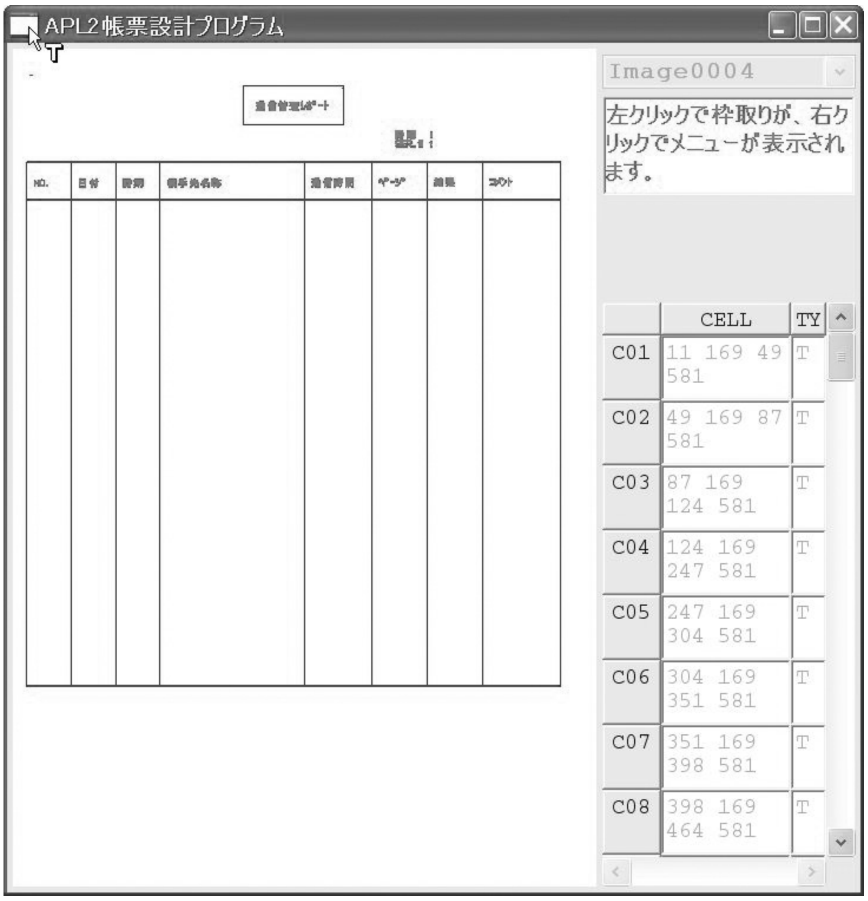
APL On | Index [0;0] | Fix time: 2008/12/22 19:29:09 | Properties: 0 0 0 0

Fig.5: `ΔDISPLAY` source code

How we handle business forms printout

Classical approaches to create business forms output in APL includes a method to draw lines with box characters as |┌└├┬┼. The same method as used in the case of `ΔDISPLAY` can be used, but for higher quality printout, positions of these block characters in the intermediate output can be used to draw lines graphically. In this case texts must also be converted into graphic images.

Before Unicode, when we used double byte-code, it presented no problems because there were only two sizes with Japanese characters identified with `X'8x'`, `X'9x'`, `X'Ex'` and `X'Fx'` codes in the first byte having twice the width of other ASCII characters.


APL2帳票設計プログラム

通信管理レポート

NO.	日付	時刻	相手先名称	通信時間	ページ	結果	コメント

Image0004

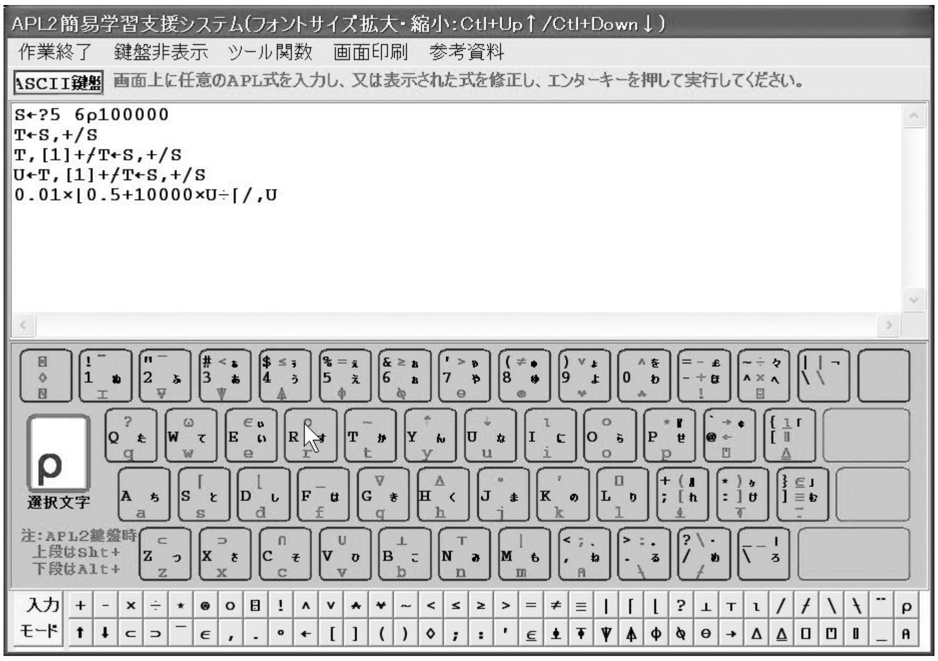
左クリックで枠取りが、右クリックでメニューが表示されます。

	CELL	TY
C01	11 169 49 581	T
C02	49 169 87 581	T
C03	87 169 124 581	T
C04	124 169 247 581	T
C05	247 169 304 581	T
C06	304 169 351 581	T
C07	351 169 398 581	T
C08	398 169 464 581	T

Fig.6: Designing business forms

How to bring new programmers to APL

Unicode has brought about an ideal environment for casual as well as professional programmers to try and learn APL2 in native languages. However, in reality, the initial cost of getting a product license is something beyond the budget of most of the prospective users. To invite these people into the world of APL effectively, neither academic initiative nor free trial use is adequate, because real persuasion comes from proper tools to prove the usefulness of APL and access to consistent and high-level consultancy and information.


APL2簡易学習支援システム(フォントサイズ拡大・縮小:Ctl+Up↑/Ctl+Down↓)
作業終了
鍵盤非表示
ツール関数
画面印刷
参考資料
ASCII鍵盤
選択文字
注:APL2鍵盤時 上段はSht+ 下段はAlt+
入力モード

画面上に任意のAPL式を入力し、又は表示された式を修正し、エンターキーを押して実行してください。

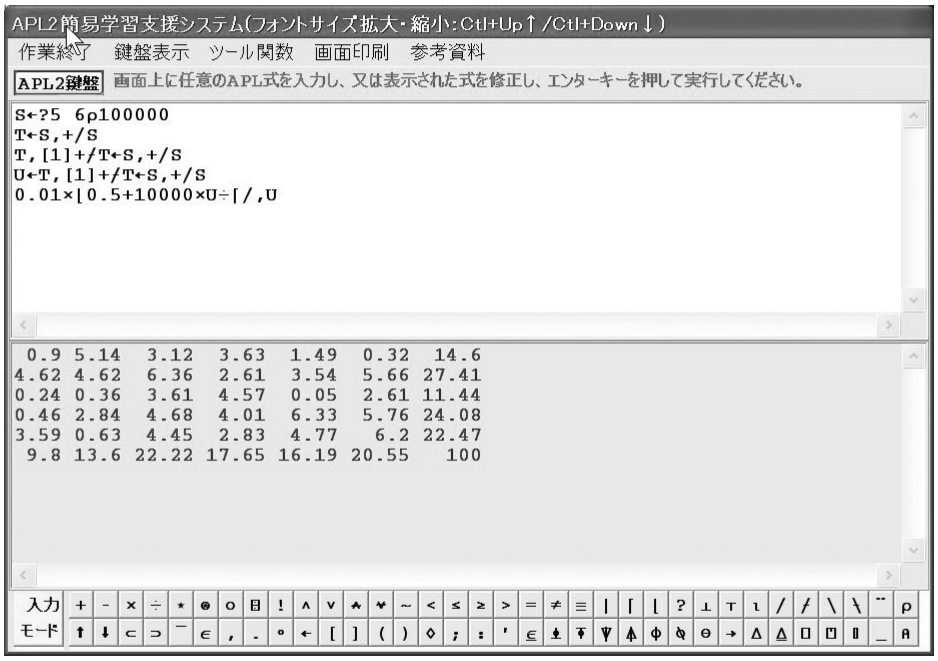
```
S←?5 6ρ100000
T←S,+/S
T,[1]+⌿T←S,+/S
U←T,[1]+⌿T←S,+/S
0.01×⌊0.5+10000×U÷⌈/,U
```

Fig.7: Japanese APL Learning System (1)

Therefore we created an APL2 package called the Japanese APL2 Learning System as shown above, which offers an inexpensive APL programming environment with full interpreter capabilities of the free IBM Workstation APL2 V2 runtime modules. It uses currently most advanced features of the Workstation APL2 V2 such as calling APL2 interpreter under application APL2 to isolate interpreter environment to avoid crashes between user entered names and the names uses in the application system program.

This package, revised totally in November 2008, supports only the Japanese users with online APL references all in Japanese at this time, and is downloadable free upon registration of e-mail address from our website aplcons.com without obligation to pay any fee for the entire system, although it is priced to cover development and support cost. It interprets every line user enters in the upper section of the window by way of ⎕EC and the result or error message will be displayed in the lower section.

It is safe from system program crashes resulting from invalid entries, because all the errors are trapped and shown in the lower section of the window. Japanese texts are directly handled in these windows as well. In addition to supporting hardware keyboard, APL-Japanese software keyboard is also provided.


APL2簡易学習支援システム(フォントサイズ拡大・縮小:Ctl+Up↑/Ctl+Down↓)
作業終了
鍵盤表示
ツール関数
画面印刷
参考資料
APL2鍵盤
入力モード

画面上に任意のAPL式を入力し、又は表示された式を修正し、エンターキーを押して実行してください。

```
S←?5 6ρ100000
T←S,+/S
T,[1]+⌿T←S,+/S
U←T,[1]+⌿T←S,+/S
0.01×⌊0.5+10000×U÷⌈/,U
```

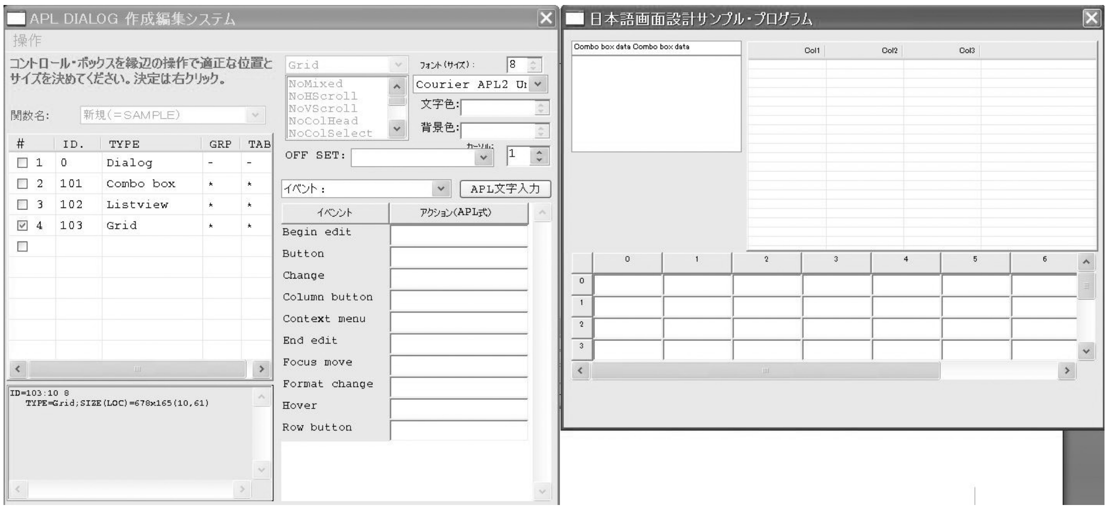
```
 0.9 5.14  3.12  3.63  1.49  0.32  14.6
4.62 4.62  6.36  2.61  3.54  5.66 27.41
0.24 0.36  3.61  4.57  0.05  2.61 11.44
0.46 2.84  4.68  4.01  6.33  5.76 24.08
3.59 0.63  4.45  2.83  4.77   6.2 22.47
 9.8 13.6 22.22 17.65 16.19 20.55   100
```

Fig.8: Japanese APL Learning System (2)

This package allows defining, editing, executing APL functions including the ones transferred from APL product environments as well. In another word, it is designed to fulfil the requirements of most of the prospective APL users to assess APL2's total capability.

Dialogue editor with APL function-generation capability

In order to encourage novice programmers to use designed dialogue windows instead of bare-input and output on the APL session manager, we developed a system to help write APL applications with embedded dialogue windows.

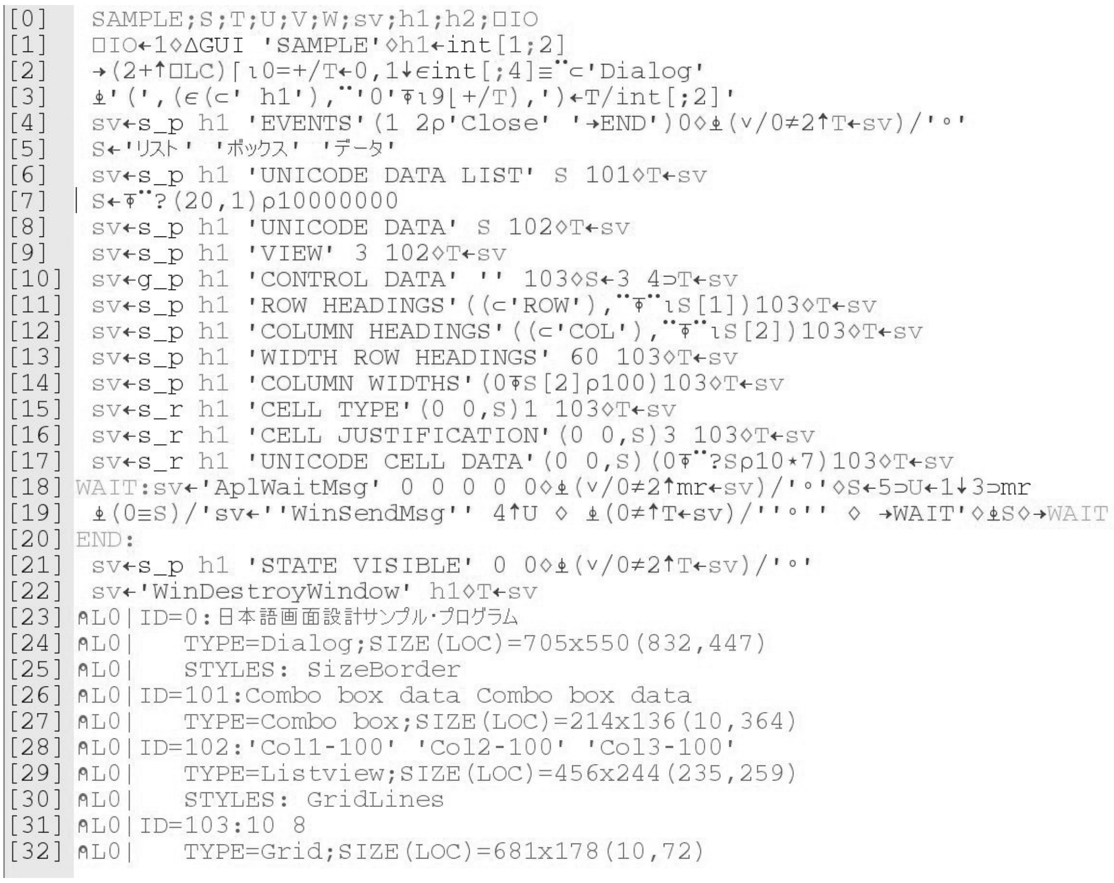


コントロール・ボックスを縁辺の操作で適正な位置とサイズを決めてください。決定は右クリック。

#	ID.	TYPE	GRP	TAB
☐ 1	0	Dialog	-	-
☐ 2	101	Combo box	*	*
☐ 3	102	Listview	*	*
☑ 4	103	Grid	*	*
☐				

```
ID=103:10 8
  TYPE=Grid;SIZE(LOC)=678x165(10,61)
```

イベント	アクション(APL式)
Begin edit	
Button	
Change	
Column button	
Context menu	
End edit	
Focus move	
Format change	
Hover	
Row button	



```
 SAMPLE;S;T;U;V;W;sv;h1;h2;⎕IO
 ⎕IO←1⋄∆GUI 'SAMPLE'⋄h1←int[1;2]
 →(2+↑⎕LC)⌈⍳0=+/T←0,1↓∊int[;4]≡¨⊂'Dialog'
 ⍎'(',(∊(⊂' h1'),¨'0'⍕⍳9⌊+/T),')←T/int[;2]'
 sv←s_p h1 'EVENTS'(1 2⍴'Close' '→END')0⋄⍎(∨/0≠2↑T←sv)/'∘'
 S←'リスト' 'ボックス' 'データ'
 sv←s_p h1 'UNICODE DATA LIST' S 101⋄T←sv
 S←⍕¨?(20,1)⍴10000000
 sv←s_p h1 'UNICODE DATA' S 102⋄T←sv
 sv←s_p h1 'VIEW' 3 102⋄T←sv
 sv←g_p h1 'CONTROL DATA' '' 103⋄S←3 4⊃T←sv
 sv←s_p h1 'ROW HEADINGS'((⊂'ROW'),¨⍕¨⍳S[1])103⋄T←sv
 sv←s_p h1 'COLUMN HEADINGS'((⊂'COL'),¨⍕¨⍳S[2])103⋄T←sv
 sv←s_p h1 'WIDTH ROW HEADINGS' 60 103⋄T←sv
 sv←s_p h1 'COLUMN WIDTHS'(0⍕S[2]⍴100)103⋄T←sv
 sv←s_r h1 'CELL TYPE'(0 0,S)1 103⋄T←sv
 sv←s_r h1 'CELL JUSTIFICATION'(0 0,S)3 103⋄T←sv
 sv←s_r h1 'UNICODE CELL DATA'(0 0,S)(0⍕¨?S⍴10*7)103⋄T←sv
WAIT:sv←'AplWaitMsg' 0 0 0 0 0⋄⍎(∨/0≠2↑mr←sv)/'∘'⋄S←5⊃U←1↓3⊃mr
 ⍎(0≡S)/'sv←''WinSendMsg'' 4↑U ⋄ ⍎(0≠↑T←sv)/''∘'' ⋄ →WAIT'⋄⍎S⋄→WAIT
END:
 sv←s_p h1 'STATE VISIBLE' 0 0⋄⍎(∨/0≠2↑T←sv)/'∘'
 sv←'WinDestroyWindow' h1⋄T←sv
⍝L0|ID=0:日本語画面設計サンプル・プログラム
⍝L0|   TYPE=Dialog;SIZE(LOC)=705x550(832,447)
⍝L0|   STYLES: SizeBorder
⍝L0|ID=101:Combo box data Combo box data
⍝L0|   TYPE=Combo box;SIZE(LOC)=214x136(10,364)
⍝L0|ID=102:'Col1-100' 'Col2-100' 'Col3-100'
⍝L0|   TYPE=Listview;SIZE(LOC)=456x244(235,259)
⍝L0|   STYLES: GridLines
⍝L0|ID=103:10 8
⍝L0|   TYPE=Grid;SIZE(LOC)=681x178(10,72)
```

Fig.9: GUI code generation

The generated APL function prototypes as shown above can be edited to develop real APL applications in a short time. A Japanese online text book explains the rules and syntax of this method (which is somewhat different from what the IBM product offers) accompanies this program.

In conclusion

Adoption of Unicode in APL which supports native languages in any part of the world can provide a good opportunity to spread the use of APL as a major programming language.

I firmly believe that this language is no longer a replacement of desk calculators. Therefore I wish the vendors of APL would emphasise the runtime with a high degree of error handling and maintenance capabilities in the language system.

References

1. “Use of APL in Japan”, Kyosuke Saigusa, Proceedings of APL94, Antwerp, ACM-SIGAPL
2. “Japanese Language Handling in APL environments”, Kyosuke Saigusa, Proceedings of APL 98, Rome, ACM-SIGAPL
3. “Simulated APL Session Manager”, Kyosuke Saigusa, Proceedings of APL2003, San Diego, ACM-SIGAPL

4. “A Simple APL-Excel Interface”, Kyosuke Saigusa, APL Quote Quad Vol.34, No.3, Summer 2004

Related articles

- “Unicode support for APL”, Morten Kromberg, 24:1, p.52
- “Review: The Solid Form of Language (Robert Bringhurst)”, Stephen Taylor, 21:2, p.44

L E A R N

Introduction to cryptography in Dyalog APL and .Net

by Dan Baronet (danb@dyalog.com)

Cryptography and how to use it in Dyalog APL and .Net. This is only an introduction to the subject. The functions presented here are only to play with.

Cryptography deals with Encryption and Decryption. Encryption is the process of converting ordinary data (*plaintext*) into unreadable form (*ciphertext*). Decryption is the reverse. A *cipher*is a pair of algorithms to create the encryption and the decryption. The algorithms are controlled by a *key* or by a pair of *keys*. Ciphers without a key can be easily broken (knowing the cipher) so keys are very important. Ciphers can be used directly for encryption or decryption without additional procedures but it is a good idea to add authentication or integrity checks to make them more secure. Another category of algorithms, used for hashing, do not have a key. All these algorithms are the primitives of cryptography.

Symmetric-key algorithms

Symmetric-key cryptography refers to encryption methods in which both the sender and receiver share the same key. They come in several flavours, some with better security in one aspect or another than others. There are two main types: block ciphers and stream ciphers.

Block ciphers work in sections (block) and include the Data Encryption Standard (DES) and the Advanced Encryption Standard (AES), designs which have been designated standards by some governments. Despite its deprecation, DES and the more secure triple-DES variant remain quite popular and are used across a wide range of applications. Many other such ciphers have been designed and released, with considerable variation in quality. Many have been cracked.

These block cipher algorithms can directly be used for the encryption and decryption of a data block without additional procedures. However, this is usually not advisable, because identical plaintext blocks encrypt into identical ciphertext blocks, out of which attackers might draw conclusions. To avoid this, these algorithms are used in operation modes, which is somewhat similar to the use of APL primitives in APL operators. Padding schemes can be used on the last block to allow the encryption of plaintexts not matching a multiple of the block size.

Other modes convert a block cipher into a stream cipher. These can continuously encrypt data on a byte or even a bit basis. It is common to all these modes to require (besides the key) an initialization vector (IV). This is a randomly chosen byte sequence in the length of a block.

Symmetric-key systems use the same key for encryption and decryption. A drawback of symmetric ciphers is the key management necessary to use them securely. Each distinct pair of communicating parties must, ideally, share a different key, and perhaps each ciphertext exchanged as well. The number of keys required increases as the square of the number of network members, which very quickly requires complex key management schemes to keep them all straight and secret. The difficulty of securely establishing a secret key between two communicating parties, when a secure channel doesn't already exist between them, also presents a chicken-and-egg problem. So, in practice, most uses have a short life cycle.

Asymmetric or public-key cryptography

In 1976 was proposed the notion of public-key cryptography in which two different but mathematically related keys are used: a public key and a private key.

In this system, the public key may be freely distributed, while its paired private key must remain secret. The public key is typically used for encryption only, while the private or secret key is used for decryption.

In addition to encryption, public-key cryptography can be used to implement digital signature schemes. A digital signature is reminiscent of an ordinary signature; they both have the characteristic that they are easy for a user to produce, but difficult for anyone else to forge. Digital signatures can also be permanently tied to the content of the message being signed; they cannot then be 'moved' from one document to another, for any attempt will be detectable. In digital signature schemes, there are two algorithms: one for signing, in which a secret key is used to process the message (or a hash of the message, or both), and one for verification, in which the matching public key is used with the message to check the validity of the signature. RSA and DSA are two of the most popular digital signature schemes.

Most public-key algorithms involve operations which are much more computationally expensive than the techniques used in most block ciphers, especially with typical key sizes. As a result, public-key cryptosystems are commonly *hybrid* cryptosystems, in which a fast high-quality symmetric-key encryption algorithm is used for the message itself, while the relevant symmetric key is sent with the message, but encrypted using a public-key algorithm. Similarly, hybrid signature schemes are often used, in which a cryptographic hash function (see below) is computed, and only the resulting hash is digitally signed.

Hash algorithms

Cryptographic hash functions are a third type of cryptographic algorithm. They take a message of any length as input, and output a short, fixed length hash which can be used e.g. as a digital signature. MD4 is a long-used hash function which is now cracked; MD5, a strengthened variant of MD4, is also widely used but cracked in practice. SHA-0 was a flawed algorithm that was withdrawn; SHA-1, now considered weak, is widely deployed and more secure than MD5, the SHA-2 family improves on it, and there's SHA-3 on the way.

Combining cryptographic primitives

By themselves, cryptographic primitives are quite limited. For instance, a bare encryption algorithm will provide no authentication mechanism, nor any explicit message integrity checking. Only when combined in security protocols, can more than one security requirement be addressed. For example, to transmit a message that is not only encoded but also protected from tinkering (i.e. it is confidential and integrity-protected), an encoding routine, such as DES, and a hash-routine such as SHA-1 can be used in combination. If the attacker does not know the encryption key, he cannot modify the message so that message hashed values can't be successfully faked.

Example of asymmetric algorithm: RSA

The RSA algorithm is among the most widely used. It involves three steps: key generation, encryption and decryption.

The following uses the notion of congruence. Two integers a and b are said to be congruent modulo n, if their difference a − b is an integer multiple of n, i.e. they have the same remainder.

Key generation

RSA is known as a deterministic encryption algorithm, i.e. it has no random component. It involves a *public* key and a *private* key. The public key can be known to everyone and is used for encrypting messages. Messages encrypted with the public key can *only* be decrypted using the private key. The keys for the RSA algorithm are generated the following way:

1. Choose two distinct prime numbers p and q.
2. Compute $n = p \times q$. n is used as the modulus for both the public and private keys.
3. Compute the totient[1] of n, $T = (p-1)\times(q-1)$.
4. Choose an integer e such that $1<e < T$, and e and T share no divisors other than 1 (i.e. e and T are *coprimes*). e is released as the public key exponent.
5. Determine d which satisfies the congruence of de and 1 modulo T. In other words, $de - 1$ can be evenly divided by the totient $T= (p - 1)x(q - 1)$. d is kept as the private key exponent.

The *public key* consists of the modulus n and the public (or encryption) exponent e. The *private key* consists of the modulus n and the private (or decryption) exponent d which must be kept secret.

Encryption

Adam transmits his public key (n,e) to Bea and keeps the private key secret. Bea then wishes to send message m[2] to Adam. She computes the ciphertext c corresponding such that c and m^e are congruent modulo n.

Bea then transmits c to Adam. If Adam's ex Eve were to *eaves*drop on the transmitted message she could make no sense of it.

Decryption

Adam can recover m from c by using his private key's exponent d like this:

We know m and c^d are congruent mod n because c^d and m^{ed} are congruent modulo n.

Since $ed = 1 + kT$, then m^{ed} is congruent modulo n, to m^{1+kT} or $m(m^T)^k$ or simply m.

The last congruence directly follows from Euler's theorem when m is relatively prime to n. It can then be shown that the equation holds for all m.

This shows that we get the original message back as c^d and m are congruent modulo n.

This is a simplified example. In reality there are all kinds of ways to complicate matters and make the life of a would-be attacker hell. For example, the choice of prime numbers should be large and the plaintext should be padded with random numbers.

A working example

Here is an example of RSA encryption and decryption in J. The parameters used here are artificially small, but one can also use OpenSSL to generate and examine a real key pair.

We'll use the verbs

```
tot =:  */ @: <:
cong=: =/@:|
NB. the (2) numbers (right arg) have the same remainder
NB. mod n (left arg)
xgcd=: 3 : 0
xy=. y|.~>/y NB. smaller first
 0 1         NB. default result
if. 0<{: t=. |/\xy do.
 'X Y'=. xgcd t
 Y, X-Y*<.%~/xy
end.
)
```

Choose two prime numbers, eg: `p =: 61` and `q =: 53`

Compute `n =: p*q NB. =3233`

Compute the totient `T =: tot p,q NB. (61-1)*(53-1) =3120`

Choose e > 1 coprime to 3120, here we pick a value not too small e = 17

Compute d such that `T cong 1,d*e` e.g., by computing the modular multiplicative inverse of e modulo T:

```
    d =:  T|{: xgcd e,T    NB. 2753
```

since 46801 = 17×2753 and 1 = 3120|46801 this is the correct answer.

The public key is (n = 3233, e = 17). The encryption statement is:

```
    c =: n|m^e
```

The private key is (n = 3233, d = 2753). The decryption statement is:

```
    m =: n|c^d
```

For example, to encrypt m = 123, we calculate

```
    c=: 3233 | 123x ^ 17    NB. 855
```

To decrypt c = 855, we calculate.

```
    m=: 3233 | 855x ^ 2753  NB. 123
```

In real life situations the primes selected would be much larger, however in our example it would be relatively trivial to factor n, 3233, obtained from the freely available public key back to the primes p and q. Given e, also from the public key, we could then compute d and so acquire the private key. The message, also, would be a much larger integer (think of a character string as a long list of bits representing a large integer).

.Net

The previous example is fairly easy to program in any language as long as the numbers are kept small. Otherwise special routines must be written to handle large numbers like the ones represented by long strings. Plus there are a number of issues that must be dealt with regarding security. Without getting into too many details let’s say that it is not easy to come up with an acceptable solution. This is where ready-made solutions like .Net come in the picture.

.Net offers a series of classes to handle cryptography. They reside in `System.Security.Cryptography` which is in `System.Core.dll`

Let's first define a few utilities, including a *hash* function:

```
:Namespace CryptoTools
    ∇ r←DNetCrypto ⍝ The location of the cryptography libraries
     ⍝ change for 64b OS:
      r←'System.Security.Cryptography,\Program Files'
      r,←'\Reference Assemblies\Microsoft\Framework\v3.5\System.Core.dll'
    ∇
    ∇ value←hash string;str;⎕USING
     ⍝ Return hash value of the string given as arg
      :If isChar str←,string
        str←⎕UCS str ⍝ turn characters into numbers for the hash fn
        {}⎕DR str    ⍝ kludge for V12 to ensure str is small int
      :EndIf
      ⎕USING←DNetCrypto                ⍝ also SHA1/384/512, CRC32
     ⍝ MD5CryptoServiceProvider:
      value←(⎕NEW SHA256Managed).ComputeHash⊂str
    ∇

    if←/⍨ ⋄ UTF8←'UTF-8'∘⎕ucs ⋄ isChar←{∨/0 2=10|⎕DR 1/⍵}
   ⍝ ⎕DR forces demotion (V12):
    UCSN←{~isChar ⍵:⍵ ⋄ v{⍺}⎕dr v←UTF8 ⍵}

:EndNamespace
```

The hash function (here SHA256, but it could be another) can be applied to any string:

```
      CryptoTools.hash  ⊃,/⎕src CryptoTools
165 157 223 218 183 249 78 22…
```

The following class will handle symmetric cryptography:

```
:Class  CodeSymmetric

  :include CryptoTools

  M2MS←{8÷⍨⍵[2]+0,s×⍳(-/2↑⍵)÷1⌈s←¯1↑⍵} ⍝ valid Key sizes [min-max]
  ∇ boa provider;choices;msg;n
  :Access public
  :Implements constructor
⍝ Data Encryption Standard (DES) supports a 64 bit key only
⍝ Rivest Cipher 2 provider supports keys from 40 to 128* bits
⍝ Rijndael (also known as AES) provider supports keys 128/192/256*
⍝ TripleDES provider (also known as 3DES) supports keys of 128/192*
  choices←'DES' 'RC2' 'Rijndael' 'TripleDES'
  msg←'Invalid provider; choose one of',⍕choices
  msg ⎕SIGNAL 99 if(⍴choices)<n←choices⍳⊂provider
  ⎕USING←DNetCrypto
  choices←DESCryptoServiceProvider RC2CryptoServiceProvider
  _algo←⎕NEW n⊃choices,RijndaelManaged
TripleDESCryptoServiceProvider

  n←⊃_algo.LegalKeySizes.(MaxSize MinSize SkipSize)
  _vks←M2MS n   ⍝ all valid Key sizes

  IV←'1Az=-@qT' ⍝ Initialisation Vector
  ⎕DF provider  ⍝ conveniently identify instance
  ∇

  ∇ r←RandomKey ⍝ This generates a random Key
    :Access public
    _algo.GenerateKey
    r←_algo.Key
  ∇

  :property Key ⍝ The key used to encrypt/decrypt data
  :access public
  ∇ r←get
    r←_key
  ∇

  ∇ set val;val;msg;t
    msg←'⍴Key must be ',((1<⍴t)/'one of '),⍕t←_vks
    msg ⎕SIGNAL 11 if~(⍴val←val.NewValue)∊t
    _algo.Key←_key←UCSN val
  ∇
   :endproperty
```

```
⍝ Using the default Cipher Block Chaining (CBC) mode, all data
⍝ blocks are processed using the value derived from the previous
⍝ block; the first data block has no previous data block to use,
⍝ so it needs an Initialisation Vector (IV) to feed the first block

 :property IV
 :access public
  ∇ r←get
    r←_iv
  ∇

  ∇ set Value;val;validBS
  ⍝ We must ensure the value fits requirements:
    validBS←M2MS⊃_algo.LegalBlockSizes.(MaxSize MinSize SkipSize)
    :If ~validBS∊⍨⍴val←Value.NewValue ⍝ if invalid block size
        val←validBS[1++/validBS<⍴val]⍴val ⍝ pick the next one
    :EndIf
    _algo.IV←_iv←UCSN val
  ∇
 :endproperty
```

```
⍝ Encrypts the specified Data using preset key and preset IV
  ∇ r←EncryptString d;⎕USING;ms;cs;cr
    :Access public
    ⎕USING,⍨←⊂'System'
   ⍝ The key and IV have better been set
    ms←⎕NEW IO.MemoryStream
    cr←_algo.CreateEncryptor ⍬
    cs←⎕NEW CryptoStream(ms cr CryptoStreamMode.Write)
    cs.Write((UCSN d)0,⍴d)
    cs.Close
    ms.Close
    r←ms.ToArray
  ∇
   ⍝ Decrypts the specified data using preset key and preset IV
  ∇ r←DecryptCipher encryptedData;b;ms;cs;len;⎕USING
    :Access public
    ⎕USING,⍨←'System' 'Dyalog'
    ms←⎕NEW IO.MemoryStream(encryptedData 0,⍴encryptedData)
    cs←⎕NEW CryptoStream(ms(_algo.CreateDecryptor
⍬)CryptoStreamMode.Read)
    b←⎕NEW ByRef,⊂⊂3/⍨⍴encryptedData
    len←cs.Read(b 0,⍴encryptedData)
    cs.Close
    r←⎕UCS len↑b.Value
  ∇
:EndClass
```

Let’s try it:

```
  s1←⎕new CodeSymmetric  'TripleDES'
  s1.Key←16⍴'secret'
  +cs1←s1.EncryptString 'Let''s rendez-vous at midnight'
155 230 195 207 193 180 216 228 2 14 11…
  s1.DecryptCipher cs1
Let's rendez-vous at midnight
```

Any instance of the class will do to decrypt the cipher as long as the Initialisation Vector and the key are the same.

Obviously, that code could be modified to, say, accept the Initialisation Vector and/or the key at instantiation time.

The following class will handle asymmetric cryptography:

```
:Class CodeAsymmetric
⍝ The only provider supported is the RSACryptoServiceProvider.
  :include CryptTools
  ⎕using← DNetCrypto ⋄ ⎕io ⎕ml←1 ⋄ ⎕wx←3
  ∇ boa0 ⍝ The public Key is automatically generated
    :Implements constructor
    :Access public
    _rsa←⎕NEW RSACryptoServiceProvider
    GenerateNewKeys
  ∇

  ∇ boa1 arg ⍝ This is where you make the public Key.
   ⍝ It could be made of INI files, XML, etc.
   ⍝ Here we only accept XML strings.
    :Implements constructor
    :Access public
    _rsa←⎕NEW RSACryptoServiceProvider
    _rsa.FromXmlString⊂arg
  ∇

⍝ Generates a new public/private key pair as strings
  ∇ {(publicKeyXML privateKeyXML)}←GenerateNewKeys
   ⍝ Generate new keys for this instance
    :Access public
    publicKeyXML←_rsa.ToXmlString 0
    privateKeyXML←_rsa.ToXmlString 1
  ∇

  ∇ r←Decrypt cipher
    :Access public
    r←UTF8 _rsa.Decrypt cipher 0
  ∇

  ∇ r←Encrypt d
    :Access public
    r←_rsa.Encrypt((UCSN d)0)
  ∇
:EndClass
```

Now, Adam creates an object with both public and private keys:

```
      adam←⎕new CodeAsymmetric
      (pub  pvt)← adam.GenerateNewKeys
```

Adam forwards the public key to Bea who uses it to send him messages:

```
      bea←⎕new  CodeAsymmetric   pub
      ⍴msg←bea.Encrypt  'meet me at midnight'
 89 12 181 136 53 …
```

Adam decrypts the message like this:

```
      adam.Decrypt  msg
 meet me at midnight
```

That’s how easy it is. Because asymmetric cryptography is calculation-intensive it is best to limit the material to small strings.

Now, in real life messages tend to be longer and symmetric cryptography works best. What some people do is to encrypt the large text symmetrically with a key which is encrypted asymmetrically. They often also add a signature, for example the hash of the message and/or encrypted key to ensure everything is kosher. Something along the lines of

```
      beamsg←1000⍴'long message... '
      code←⎕new CodeSymmetric 'RC2'
      code.Key←cpw←'secret'
      ⍴cryptedmsg←code.EncryptString beamsg
 1008
      ⍴cryptedkey←bea.Encrypt cpw
 128
      ⍴H←CryptoTools.hash cryptedkey,cryptedmsg
 32
```

She then sends Adam the encrypted message, the encrypted key and the hash. Adam first checks there's been no tampering:

```
      H≡CryptoTools.hash cryptedkey,cryptedmsg
 1
```

He then finds the key used to encrypt the message:

```
      adam.Decrypt cryptedkey
 secret
```

Then he finally decodes the message:

```
      decode←⎕new CodeSymmetric 'RC2'
      decode.Key←'secret'
      decode.DecryptCipher cryptedmsg
 long message... long message... long message...
```

That is the basis of message exchange protocols like SSL which will bundle up all the stuff to transfer, adding their own packaging information.

Epilogue

The topic of cryptography is fairly complex. There are many issues related to this which are out of the scope of this article.

Feel free to play with this; there are a number of methods to en/decrypt streams (files) and others. Have a look.

Notes

1. The totient (usually denoted φ) of a number is defined as the number of coprimes of that number. For a prime number P it is P-1.
2. Here the Message is turned into an (large) integer using a technique known as "padding" which is irrelevant to the description

FINANCIAL MATH IN Q

3: The annuity

by Jan Karman (jkarman@planet.nl)

The annuity differs from the other applications in that it doesn’t need external files. It is simply (and simple) mathematics. The program produces a survey of the amortisation of an annuity loan, with the necessary controls. For convenience we shall define some auxiliary functions so that the development of the annuity function will be very easy, almost trivial. So, the purpose is just to show what is possible with a few lines of K. The annuity loan is in wide use for mortgages and a large scale of many other types of private loans.

Mathematics

In actuarial practice every value is being brought back to the point of time 0 – that’s where we are – usually called “the present value”. The present value of an annuity, a financial form in which a unit of capital is being paid at the end of every consecutive year, is denoted by a_n. If an individual wants to settle for a loan and repaying it by way of a yearly level amount t, then $t.a_n$ should be 1, according to the equivalence principle as the axiom of all financial theory, and $t = 1/a_n$. If i is the interest rate and v denotes the present value of the unit 1 at the end of the year $v = 1/1 + i$.

The present value of each payment in the annuity can be written as

$v^1, v^2 \dots v^{(n-1)}, v^n$

and the total value of the annuity

$a_n = v^1 + v^2 + \dots + v^{(n-1)} + v^n$ **(1)**

Multiplied by the ratio (1+i) we get

$(1+i).a_n = v^0 + v^1 + \dots + v^{(n-2)} + v^{(n-1)}$ **(2)**

and like an ordinary geometric series we subtract (1) from (2)

$i.a_n = 1 - v^n$

getting

$a_n = (1-v^n)/i$

The level annual payment of t contains two components: an interest component and a redemption component. Two successive outstanding balances amount to $t.a_{n-m}$ and $t.a_{n-m+1}$ and therefore the debt appears to be decreased with $t.v^{n-m+1}$ immediately after the m^th^payment.

A little detour

It may seem all trivial, but the geometric series has its dark and obscure caverns. We have seen that the yearly payments have two components, an interest and a redemption part. Now, the first redemption, a_1, equals to the level payment, t, minus the indebted interest i, being t-i. In the second year this component increases to (t-i).(1+i), and all the remaining redemption parts form again a geometrical series with ratio (1+i), adding up, of course, to the initial amount of the debt 1, and the series will be cumulated to s_n, the accumulated value of an annuity – also known as a saving contract.

Thus $(t - i).s_n = 1$, so $t - i = 1/s_n$.

Therefore, since by definition

$t = 1/a_n$

it follows that

$1/a_n - i = 1/s_n$

and

$1/a_n - 1/s_n = i$.

Here we see the remarkable relationship between the present value and the accumulated value of an annuity on the one hand and the interest rate on the other. Indeed, the difference between the reciprocal of the present value of an annuity and the reciprocal of the accumulated value of the same annuity results in the basic ingredient: the interest rate. Of course we could come to this result by reducing right from the definitions – that would give a longer detour.

Back to business.

Effective interest rate

An interest rate is most typically quoted as an annual percentage. In practice, however, interest rates are being paid in fractions, say half-yearly, quarterly or monthly or even in days. In theory it can be paid in infinitely small fractions, in which case we have to do with continuous interest rates. Here, we will confine to the discrete method.

It would be logical when calculating a monthly interest that the accumulated monthly fractions would equal the contractual interest. Thus that j in

$j = ((1 + i'/12)^{12}) – 1$

would equal i, with of course i' somewhat lower than i.

Banks know better, they are the money experts and they calculate

$j = ((1 + i/12)^{12}) – 1$

(Once, when I showed this to my brother in law, he checked his mortgage contract, which stated an “annual interest rate of 6%, payable monthly” – he lodged a complaint with his bank, with success).

The K-implementation

```
/Global functions
f:{100*((1+0.01*x%y)^y)-1}    / real interest
vn:{(1+0.01*z%y)^-x*y}        / present value
an:{(1-vn[x;y;z])%0.01*z%y}   / annuity
sumrnd:{x*(*t)-':t:_.5++\y%x} / rounding function
```

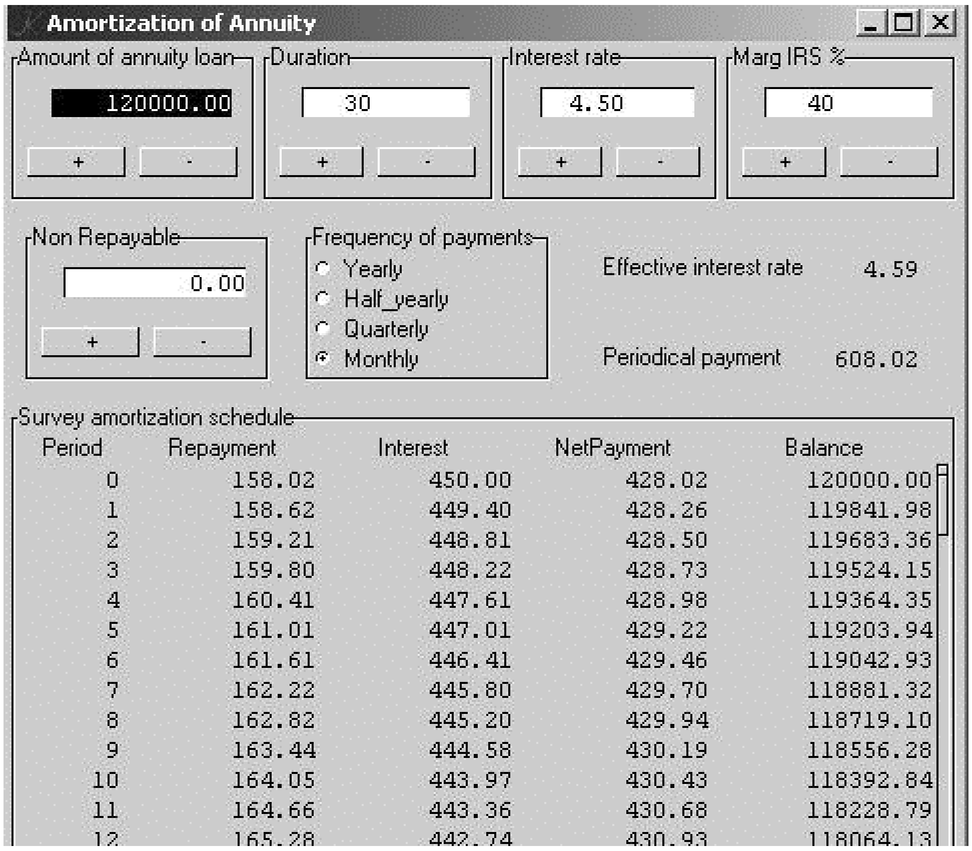
Calculations

The calculations are all being done in one dependency:

```
t..d:".$term"
eff..d:"f[I.i;t]"
eff..f:5.2$
ann..d:"B.bd*%an[D.d;t;I.i]"
ann..f:8.2$
tm..d:"!D.d*t"
at..d:"sumrnd[0.01;(ann-B.bd*0.01*I.i%t)*(1+0.01*I.i%t)^!D.d*t]"
it..d:"(nR.nbd*0.01*I.i%t)+ann-at"
ml..d:"at+(1-0.01*IB.ib)*it"
rs..d:"nR.nbd+B.bd-+\\0,-1 _ at"
```

Picture

… and the `show` gives this:



Amount of annuity loan: 120000.00
Duration: 30
Interest rate: 4.50
Marg IRS %: 40
Non Repayable: 0.00

Frequency of payments
Yearly [ ] Half_yearly [ ] Quarterly [✓ Monthly]

Effective interest rate 4.59
Periodical payment 608.02

Survey amortization schedule

Period	Repayment	Interest	NetPayment	Balance
0	158.02	450.00	428.02	120000.00
1	158.62	449.40	428.26	119841.98
2	159.21	448.81	428.50	119683.36
3	159.80	448.22	428.73	119524.15
4	160.41	447.61	428.98	119364.35
5	161.01	447.01	429.22	119203.94
6	161.61	446.41	429.46	119042.93
7	162.22	445.80	429.70	118881.32
8	162.82	445.20	429.94	118719.10
9	163.44	444.58	430.19	118556.28
10	164.05	443.97	430.43	118392.84
11	164.66	443.36	430.68	118228.79
12	165.28	442.74	430.93	118064.13

The complete application is available online and can be downloaded freely from http://www.ganuenta.com/annuity_k.exe.

There is also an APL version at http://www.ganuenta.com/annuity.exe built in Dyalog APL by means of Causeway. From this version neat reports can be printed by use of Newleaf, Adrian Smith's DTP application – so, 100% APL.

Appendix

(May be downloaded from
http://archive.vector.org.uk/content/published/karman/annuity.k)

```
/ Amortization scheme for annuity
/ Variables: amount(bd), duration(d), interest rate(i), marg
IRS(IB)
 \m f courier new-9
/ \m l arial-8
  \m c 0 -1 808080
  \p 16
  \c 0
```

```
/Global functions
f:{100*((1+0.01*x%y)^y)-1}/ real interest
vn:{(1+0.01*z%y)^-x*y}/ present value
an:{(1-vn[x;y;z])%0.01*z%y}/ annuity

/ Dictionaries
\d .k.B
bd: 1.0*120000; bd..l:""; bd..f:12.2$
incbd:"bd+:1000"; decbd:"bd-:1000"
incbd..c:decbd..c:`button
incbd..l:"+"; decbd..l:"-"
incbd..f:16.2$
.k.B..l:"Amount of annuity loan"
.k.B..a:(`bd;`incbd`decbd)

\d .k.nR
nbd: 1.0*0; nbd..l:""; nbd..f:12.2$
incnbd:"nbd+:1000"; decnbd:"nbd-:1000"
incnbd..c:decnbd..c:`button
incnbd..l:"+"; decnbd..l:"-"
.k.nR..l:"Non Repayable"
.k.nR.[`x]:12
.k.nR..a:(`nbd;`incnbd`decnbd)

\d .k.D
d: 30; d..f:4$; d..l:""
incd:"d+:1"; decd:"d-:1"
incd..c:decd..c:`button
incd..l:"+"; decd..l:"-"
.k.D..l:"Duration"
.k.D..a:(`d;`incd`decd)

\d .k.I
i:4.50; i..f:5.2$; i..l:""
inci:"i+:0.01"; deci:"i-:0.01"
inci..c:deci..c:`button
inci..l:"+"; deci..l:"-"
.k.I..l:"Interest rate"
.k.I..a:(`i;`inci`deci)

/In some countries interest paid on a (mortgage) loan is
/deductible from income for IRS;
```

```
\d .k.IB
ib:40; ib..f:4$; ib..l:""
incib:"ib+:1"; decib:"ib-:1"
incib..c:decib..c:`button
incib..l:"+"; decib..l:"-"
.k.IB..l:"Marg IRS %"
.k.IB..a:(`ib;`incib`decib)

\d ^

eff..e:ann..e:0
eff..l:"    Effective interest rate    "
ann..l:"    Periodical payment"

Yearly:1; Half_yearly:2; Quarterly:4; Monthly:12
term:`Monthly
term..l:"Frequency of payments"
term..c:`radio
term..o:(`Yearly `Half_yearly `Quarterly `Monthly)
term..x:18

t..d:".$term"
eff..d:"f[I.i;t]"
eff..f:5.2$
ann..d:"B.bd*%an[D.d;t;I.i]"
ann..f:8.2$
tm..d:"!D.d*t"
at..d:"sumrnd[0.01;(ann-B.bd*0.01*I.i%t)*(1+0.01*I.i%t)^!D.d*t]"
it..d:"(nR.nbd*0.01*I.i%t)+ann-at"
ml..d:"at+(1-0.01*IB.ib)*it"
rs..d:"nR.nbd+B.bd-+\\0,-1 _ at"

/ Survey
hdr:`Period`Repayment`Interest`NetPayment`Balance
fs:("6$.k.tm";"13.2$.k.at";"13.2$.k.it";"13.2$.k.ml";"15.2$.k.rs")
comp:{[x;y;z]
      a:.,(`e;0)
      t:.+(x;y;a)
      .[t;(~x;`d);:;z]}
Survey:comp[hdr;&#hdr;fs]
Survey..l:"Survey amortization schedule"

.k..l:"Amortization of Annuity"
.k..a:((`B`D`I`IB);(`nR;`term;`eff`ann);`Survey) / rearrange
display
```

```
`show$`.k
/=================================================================
\
Description: This program produces a survey of the amortization of
an annuity loan.
In the top section of the screen are four controls for the data.
In the middle section are two controls and display of real interest
and yearly annuity.
The bottom section shows the amortization scheme with one line for
every payment, giving redemption part, interest part, net cost and
balance.
The + and - buttons are supposed to behave like spinboxes.

\
 Comments & questions welcome
 Middelburg (Neth), January 2006
 info@ganuenta.com
```

NO STINKING LOOPS

Tables with calculated columns

by Stevan Apter

This article is the second in an occasional column, *No Stinking Loops*. Stevan Apter is one of the programmers Jeffry Borror referred to as “the q gods” in his textbook *q for Mortals*.

K4 has a quite different way of representing tables from that used in K3. This article describes how to simulate in q the column dependencies that K3 supported.

0. Dependencies

In K3, tables are pseudotypes, dictionaries of vectors. The ‘columns’ of a ‘table’ are first-class variables. Thus, using dot-notation:

```
   t.f:10 20 30
   t g:40 50 60
```

The ‘records’ of a table are the corresponding elements of the ‘column’ variables.

A variable is a data-structure with three components:

- a simple symbolic name, e.g. `` `f ``
- a value, e.g. `10 20 30`
- a recursive dictionary of attributes

The `d` attribute is used to define the variable as a functional dependency. For example, using double-dot-notation:

```
   t.h..d:"f+g"
   t.h
50 70 90
```

Thus, `h` is a variable in the dictionary `t`, having name `t`, value `50 70 90`, and an attribute dictionary containing a single variable with name `d`, value `"f+g"`, and empty attribute dictionary:

```
   T
 .((`f
    10 20 30
    )
   (`g
    40 50 60
    )
   (`h
    50 70 90
    .,(`d;"f+g";)))
```

The evaluation of `t` recursively evaluates the columns `f`, `g`, and `h` of `t`. `t.h` depends on `f` and `g`. If either `f` or `g` changes, `h` is marked 'invalid'. If `h` is invalid, then on reference it recalculates in `t` using `f` and `g`.

The K3 workspace is a tree of dictionaries rooted in the nameless dictionary. We can capture the entire workspace by evaluating the empty symbol; or, as one wag put it, "The value of nothing is everything":

```
   .`
 .((`k
    .,(`t
         .((`f
              10 20 30
              )
             (`g
              40 50 60
              )
             (`h
              50 70 90
              .,(`d;"f+g";)))
         )
    )
   (`t;-7.584544e+008;))
```

In q (i.e. K4) a table is a list of records – structurally identical dictionaries. The columns of a table are the corresponding entries of the records. (N.B. internally, q stores the table as a structure of column-lists.)

How then can we simulate in q the column dependencies we get for free in K3?

1. Tables and views.

In q we have tables:

```
q)t:([]f:10 20 30;g:40 50 60)
q)t
f  g
-----
10 40
20 50
30 60
```

and we have views:

```
q)v::select from t where g<60
q)v
f  g
-----
10 40
20 50
```

`v` depends on `t`. When `t` is modified, `v` becomes ‘invalid’. `V` will be recalculated (‘validated’) the next time it is referenced:

```
q)t+:10
q)t
f  g
-----
20 50
30 60
40 70


q)v
f  g
-----
20 50
```

2. Calculated columns

We can use `update` to add calculated columns to a table:

```
q)update k:f+g,l:neg f from t
f  g  k   l
-------------
20 50 70  -20
30 60 90  -30
40 70 110 -40
```

but q won't allow us to add columns unless they depend strictly on existing columns in the table:

```
q)update k:f+g,l:k*2 from t
'k
q)update l:k*2,k:f+g from t
'k
```

Columns must be added as a correctly-ordered sequences of separate updates:

```
q)update l:k*2 from update k:f+g from t
f  g  k   l
-------------
20 50 70  140
30 60 90  180
40 70 110 220
```

3. Automating calculated columns

Let's divide up the parameters to the problem and assign them to distinct variables:

`t`	is a table
`f`	is a data-structure containing the names and definitions of calculated columns
`v`	is a view which depends on `t` and `f`

For example, `f` might be defined through a GUI, in which users specify the calculated columns.

`v` will be a view which functionally depends on `t` and `f`:

```
v::willbe[t;f]
```

The function `willbe` takes `t` and `f` as arguments and returns a table containing the columns of `t` plus the defined columns, or 'willbes', specified by `f`.

4. Implementation 1: update over

To start, let's use q's native parsing primitive to analyse the structure of the successful update from section 2:

```
q)parse"update l:k*2 from update k:f+g from t"
!
(!;`t;();0b;(,`k)!,(+;`f;`g))
()
0b
(,`l)!,(*;`k;2)
```

The functional form of `update` is:

```
![t;a;b;c]
```

where `t` is the table to be updated, `a` is the constraint, or 'where' clause, `b` is the group, or 'by' clause, and `c` is the dictionary of names and definitions of the columns to be added.

In the problem at hand we have no constraints and no grouping, so by convention `a` and `b` are `()` and `0b`.

The column definitions are unit dictionaries. Each one gives the name and definition of a single field. A unit dictionary maps a one-element symbolic vector – the name – to a one-element parse of the definition. So, for example:

```
q)enlist[`k]!enlist parse"f+g"
k| + `f `g
```

The form of the successful update is:

```
![![t;();0b;kdict];();0b;ldict]
```

Where `kdict` and `ldict` are unit-dictionaries which define `k` and `l` respectively. Generalising, we see that in order to add columns `c1…cn` to `t` we have to construct an expression of the form:

```
![..![t;();0b;c1]..;();0b;cn]
```

Let's simplify this by defining a function which eliminates the constants:

```
col:{![x;();0b;enlist[y]!enlist z]}
```

`col` takes three arguments: `x` is a table, `y` is a symbol, and `z` is the parse of a definition. `Col` returns `x` updated with the new column `y`. So:

```
col[..col[t;`c1;def1]..;`cn;defn]
```

We know what to do with this pattern: express it as the application of `col` over `t`, a vector of names, and a list of definitions:

```
col/[t;`c1..`cn;(def1;..;defn)]
```

`col` executes n times. Initially, `t` is updated with `c1` to produce `t1`. Then `t1` is updated with `c2` to produce `t2`.

```
q)col/[t;`k`l;parse each("f+g";"k*2")]
f  g  k   l
-------------
20 50 70  140
30 60 90  180
40 70 110 220
```

5. Implementation 2: column references

q still requires that we order the definitions correctly:

```
q)col/[t;`l`k;parse each("k*2";"f+g")]
{![x;();0b;enlist[y]!enlist z]}
'k
```

Let's start by having `f`, our dictionary of column definitions:

```
q)f:`l`k!("k*2";"f+g")
q)f
l| "k*2"
k| "f+g"
```

Then parse each expression:

```
q)p:parse each f
q)p
l| * `k 2
k| + `f `g
```

We want to order `p` by column-reference, but first we have to extract these from each parsed definition. The algorithm is recursive: descend the parse tree looking for symbol atoms:

```
ref:{$[-11=t:type x;x;t;();.z.s each x]}
```

We read the conditional `$[..]` as follows:

- if `x` is a symbol, return `x`
- else if `x` is not a list, return `()`
- else `self each x`

For example:

```
q)ref p
l| () `k ()
k| () `f `g
```

Since each recursion adds a level of nesting, we need to flatten the result. The q primitive *raze*(K: `,/`) takes a list of sublists and returns the catenation of the sublists. The raze of `x` reduces one level of nesting. And since references can occur more than once in an expression, we need to compress out duplicates with `distinct`.

Since the result of `ref` is a tree it will often contain more than one level of nesting. For example:

```
q)ref parse "(a+b)*c-a*b"
()
(();`a;`b)
(();`c;(();`a;`b))
```

But `distinct raze` reduces just one level:

```
q)distinct raze ref parse "(a+b)*(a*b)-c"
()
`a
`b
(();`a;`b)
`c
```

To flatten a list of arbitrary depth, we keep applying `raze` until the result converges. That is, until the result cannot be any flatter:

```
flatten:distinct raze over

q)flatten ref parse "(a+b)*(a*b)-c"
`a`b`c
```

So the final form of our function for extracting references from a parse-tree is:

```
refs:flatten ref@
```

Thus:

```
q)refs each p
l| ,`k
k| `f`g
```

6. Implementation 3: column order

Now that we have the references of `p` just one piece remains: re-order `p` so we can add columns to `t` in correct order.

Let's define f as follows:

```
q)f:`h`j`k!("j+k";"f+g";"j*100")
q)f
h| "j+k"
j| "f+g"
k| "j*100"
q)p:parse each f
q)p
h| + `j `k
j| + `f `g
k| * `j 100
q)r:refs each p
q)r
h| `j`k
j| `f`g
k| ,`j
q)k:key r
q)k
`h`j`k
```

Now if we index `r` by `k`:

```
q)r k
`j`k
`f`g
,`j
```

we get the references of each definition. Notice that some symbols in the result are indices of `r(`h`j`k)` and some are not (`` `f`g``). Indexing `r` by one of the latter returns an empty list:

```
q)r r k
(`f`g;,`j)
(`symbol$();`symbol$())
,`f`g
q)r r r k
((`symbol$();`symbol$());,`f`g)
(();())
,(`symbol$();`symbol$())
```

In this process, we are drilling down in parallel to the ultimate constituents of each definition in `r`. Eventually, we bottom out in a nest of empties, since the ultimate constituents (`f` and `g`) are columns of `t`:

```
q)r over k
((();());,(();()))
(();())
,(();())
```

To capture the sequence of intermediates, we use `scan`:

```
q)r scan k
h                                  j                        k
`j`k                               `f`g                     ,`j
(`f`g;,`j)                         (`symbol$();`symbol$())  ,`f`g
((`symbol$();`symbol$());,`f`g)    (();())
(`symbol$();`symbol$())
((();());,(`symbol$();`symbol$())) (();())                  ,(();())
((();());,(();()))                 (();())                  ,(();())
```

Reversing the result, we get the constituent analysis of `f` in calculation order:

```
q)reverse r scan k
((();());,(();()))                 (();())                  ,(();())
((();());,(`symbol$();`symbol$())) (();())                  ,(();())
((`symbol$();`symbol$());,`f`g)    (();())
,(`symbol$();`symbol$())
(`f`g;,`j)                         (`symbol$();`symbol$())  ,`f`g
`j`k                               `f`g                     ,`j
h                                  j                        k
```

The nesting and the empties are irrelevant, so we flatten the analysis:

```
q)flatten reverse r scan k
`f`g`j`k`h
```

This gives us a valid calculation order: the ultimate constituents first (in no particular order), followed by `j (f+g)`, `k (j*100)`, and `h (j+k)`.

For complex `f`, `flatten over reverse r scan key r` produces ever-larger and more complex intermediates. By moving the flattening operation into the scan loop we keep the intermediate results as simple as possible, thereby reducing the complexity of the indexing:

```
q)(flatten r@)scan k
`h`j`k
`j`k`f`g
`f`g`j
`f`g
`symbol$()
()
q)flatten reverse(flatten r@)scan k
`f`g`j`k`h
```

Finally, we want to exclude non-calculated constituents from the result:

```
q)flatten[reverse(flatten r@)scan k]inter key k
`j`k`h
```

So our final function is:

```
order:{flatten[reverse(flatten x@)scan key x]inter key x}
```

7. Synthesis

Putting it all together:

```
willbe:{[t;f]
 p:parse each f;        / parse of expression
 r:refs each p;         / references
 o:order r;             / ordered by reference
 col/[t;o;p o]}         / create view
flatten:distinct raze over
ref:{$[-11=t:type x;x;t;();.z.s each x]}
refs:flatten ref@
col:{![x;();0b;enlist[y]!enlist z]}
order:{flatten[reverse(flatten x@)scan key x]inter key x}
```

Let’s run through an example:

```
t:([]f:10 20 30;g:40 50 60)
f:`h`j`k!("j+k";"f+g";"j*100")
v::willbe[t;f]
q)t
f  g
-----
10 40
20 50
30 60
```

```
q)v
f  g  j  k    h
------------------
10 40 50 5000 5050
20 50 70 7000 7070
30 60 90 9000 9090
q)t:update g:g+1 from t where f<50
q)t
f  g
-----
10 41
20 51
30 61
q)v
f  g  j  k    h
------------------
10 41 51 5100 5151
20 51 71 7100 7171
30 61 91 9100 9191
```

8. Partitioned calculations

Our implementation allows us to define new columns by applying their definitions to existing columns as wholes. For example, `h` is all of `j` plus all of `k`. Suppose we add a grouping column `e` to `t`:

```
q)t:([]e:1 1 2;f:10 20 30;g:40 50 60)
q)t
e f  g
-------
1 10 41
1 20 51
2 30 61
```

Then we may want to define a new column whose values are computed for each `e`-partition of `t`: for the subtable where `e=1` and the subtable where `e=2`.

Notice that our definition of `col` supplies the constant `0b` to the third position of `!`, so new columns are always computed on the ungrouped input table `x`:

```
col:{![x;();0b;enlist[y]!enlist z]}
```

Let’s add a parameter to `willbe` which controls grouping:

```
q)f:`h`j`k`l!(’j+k";"f+g";"j*100";"k%sum k")
q)g:`h`j`k`l!(0b;0b;0b;enlist[`e]!enlist`e)
```

The new parameter `g` is a dictionary of ‘group by’ clauses, corresponding to the definitions of the calculated columns `f`.

We revise our suite of functions accordingly:

```
willbe:{[t;f;g]
 p:parse each f;          / parse of expression
 r:refs each p;           / references
 o:order r;               / ordered by reference
 col/[t;g o;o map'p o]} / create view
flatten:distinct raze over
ref:{$[-11=t:type x;x;t;();.z.s each x]}
refs:flatten ref@
map:{enlist[x]!enlist y}
col:![;();;]
order:{flatten[reverse(flatten x@)scan key x]inter key x}
```

So that:

```
q)t:([]e:1 1 2;f:10 20 30;g:40 50 60)
q)f:`h`j`k`l!("j+k";"f+g";"j*100";"k%sum k")
q)g:`h`j`k`l!(0b;0b;0b;enlist[`e]!enlist`e)
q)v::willbe[t;f;g]
q)v
e f  g  j  k    h    l
------------------------------
1 10 40 50 5000 5050 0.4166667
1 20 50 70 7000 7070 0.5833333
2 30 60 90 9000 9090 1
```

You can find this source code at www.nsl.com/q/willbe.q

Acknowledgements

Thanks to Attila Vrabecz and Michael Rosenberg for corrections and improvements.

J-OTTINGS 53

j Complex? You bet!

by Norman Thomson

j doesn't necessarily mean complex!

Although j in J normally means 'complex', numbers of the form `7j2` can model other forms of duple such as odd ratios, for example `7j2` can model odds of 7 to 2 (that is 7 to 2 against), from which fractional odds (`fro`) are obtained as

```
   fro=.({: % +/)@:+."0 NB. +. transforms ajb to a b
   fro 3j1
0.25
```

`%fro` then gives what is returned (winnings and stake) after a successful unit bet. The accumulated fractional odds of a field of three in which the odds offered by a bookmaker are evens, 3-1 and 7-2 is

```
   (+/@:fro)1j1 3j1 7j2
0.9722
```

Of course (pun intended!) bookmakers and betting shops see to it that such a sum is never less than 1, the excess over 1 being what the bookmaker creams off as markup or *overround*. In practice, *real probabilities*, that is the *absolute* probabilties of the various horses winning, vary dynamically right up to the final minutes before a race. Real probabilities reflect the many technicalities of racing as a sport such as the assessment of horses, jockeys, trainers, weather, racetrack condition, even insider trading and corruption, all of which lends a certain naïvety to the fact that some of the observations on which this article is based are derived from the single sets of static odds quoted in the racing pages of daily newspapers. However, broad conclusions can be drawn, for example that in UK horse racing overround seems to average between 25% to 40%. (It goes without saying that should any reader discover a race card for which the `+/@:fro` is less than 1, he or she should immediately raise every possible penny to place bets on all horses in multiples of `fro` and even more importantly should as a matter of duty contact me urgently!)

Add a couple of horses to the above field to make matters more realistic :

```
   (+/&:fro) fld=.1j1 3j1 7j2 5j1 8j1
1.25
```

giving an overround of 25%. Thus if bets are placed in the proportions `fro fld` then the cost of betting on all horses will be 1.25 for an assured return of 1 and an assured gain to the bookmaker of 0.25.

Assuming that the bookmaker's *quoted* odds reflect his view of the *relative* probabilities of the horses winning the race, the underlying *true odds* are:

```
   to=.(%+/)&:fro       NB. true odds
   to fld
0.4 0.2 0.1778 0.1333 0.08889
```

(Technical note: the hook `%+/` normalises a list so that the total of its elements is 1.)

`To` also demonstrates the extent to which the bookmaker downgrades odds in order to achieve overround, e.g. the horse quoted at evens has in fact a probability of 0.4 of winning the race. Also the returns (that is, including the original stake) multiplied by the true odds remain the same whichever horse wins the race :

```
   (to * %@fro)fld
0.8 0.8 0.8 0.8 0.8
```

namely the reciprocal of the overround. The bookmaker accepts bets to create a *book*, on which he reckons to make the overround as profit whatever the outcome of the race.

Random real probabilities totalling 1 are generated by

```
   rnd=.?@#&0       NB. random uniforms in {0,1}
   rrp=.(%+/)@:rnd  NB. random real probabilities
   rrp 5
0.176 0.28 0.047 0.232 0.265
```

Using these and a book based on `fld`, the bookmaker's long-term income and outgoings based on horses winning with random probabilities are given by

```
   book=.40 20 18 13 9
   (+/book),+/book*(rrp 5)*%fro fld
100 79.85
```

Significant risk to the bookmaker arises only if both his book and the real probabilities change. Bookmaker's arithmetic is a continuous process with input parameters: *current book, real probabilities, current actual odds* in which he strives to adjust his quoted odds in order to keep the book in balance, and thereby his profits secure. Incomings and outgoings can be formalised in a verb whose left argument is `book;real probabilities`, and whose right argument is `current actual odds`. A balanced book would be simply a multiple of the real probabilities. The example below shows how real probabilities make little difference to the bookmaker's expectations even if public assessment of the race shifts dramatically in favour of the outsider :

```
   inout=.dyad : '(+/>{.x),+/*/(>x),%fro y'
   book_rp=.40 20 18 13 9;0.2 0.1 0.1 0.1 0.5
   book_rp inout fld
100 80.4
```

However, suppose that the outsider attracts a large number of bets :

```
   book_rp=.40 22 18 13 50;0.2 0.1 0.1 0.1 0.5
   book_rp inout fld
143 265.7
```

This gives the bookmaker a projected loss of 123. His options are (1) to sustain his previous belief in the relative probabilities but reduce exposure to the new favourite by reducing its quoted odds, in the hope that future bets on the other horses may help to recoup his losses:

```
   book_rp=.40 22 18 13 0;0.2 0.1 0.1 0.1 0.5
   book_rp inout 1j1 3j1 7j2 5j1 1j2
93 40.7
```

or (2) to accept the new real probabilities and requote all his odds based on these. The primitive verb `j.` transforms `a b` to `ajb`, that is fractions back to odds:

```
   (j./@:(%/,-.))0.78
0.78j0.22
```

(The hook `,-.` returns a fraction joined to its 1s complement)

However, it is more satisfactory to have odds in the form `1jx` or `xj1` so define

```
   odds=.monad :0
t=.%/y,1-y
if.t>1 do.r=.>.1,t
else. r=.<.(%t),1 end.
r=.j./r
)
   odds"(0)0.5 0.25
1j1 3j1
```

(Note: Any rounding favours the bookmaker which seems quite reasonable given that odds can never be a scientifically precise measure.)

The bookmaker might choose to revise his true odds and apply an overround of about 25% to give

```
   odds"(0) 0.2 0.1 0.1 0.1 0.5*1.25
3j1 7j1 7j1 7j1 1j2
```

It is not suggested that bookmakers carry out any such arithmetic formally, although the above presumably models roughly the nimble calculations which they instinctively perform.

Beating the bookie

Given the inherent bias in favour of the bookmaker, are there any ways by which the better can possibly turn the situation to his advantage? First assume that he has some technical knowledge of which he feels reasonably assured and believes to be superior to that of the bookmaker.

Since `%fro fld` gives the returns for a unit bet the returns for any list of bets are

```
    rets =.[*%@:fro@]   NB. left argument = bets
    1 1 1 1 1 rets fld
2 4 4.5 6 9
```

Suppose now that as a matter of judgement the better believes that the race will certainly be won by one of the two favourites with probabilities in proportion 3:2. His expected returns for a bet which reflects this are

```
   6 4 0 0 0 rets fld
12 16 0 0 0
```

that is, for a total outlay of 10 he will achieve a return of either 12 or 16 or 0. His expectation, using true odds, is (0.4×6) + (0.2×12) = 4.8 which would give the bookmaker an expected gain of 5.2. However the expectation based on his own judgement is (0.6×12) + (0.4×16) = 13.6, and so if he has complete confidence in his judgement and behaves rationally, it would be senseless for him *not* to bet, nor indeed would he be unhappy if one of the unbacked horses won, since he would still have achieved value for his money in the same sense that an insurance policy on which no claim is made has nevertheless provided valuable cover.

Alternatively the better might choose to use the judgement of others, e.g. newspaper tipsters. What are the net gains or losses resulting from a unit bet on every tipster recommendation for a given day? On a day in which 20 races were run and four winners were tipped at 4-1, 11-4, 7-2 and 4-1, the net gain achieved for unit bets placed by following a tipster was given by :

```
   tips=.-~+/@:(1&rets)@ ]
   20 tips 4j1 11j4 7j2 4j1
_1.75
```

that is an overall loss of -1.75. Empirical evidence using the racing correspondents of the *Times* and the *Daily Telegraph* shows that following tipsters' advice consistently is very rarely profitable, and even then only when a winner happens to be picked at unusually long odds.

Turn now to manipulating probabilities, are there any techniques based on probability alone which can swing the bias away from the bookmaker towards the better? Such a possibility is demonstrated by the so-called Martingale in which a stake is progressively doubled for a losing bet and betting stops on a winning one. In a fair game at evens, e.g. coin tossing with bets on a tail, a tail is bound to occur eventually, at which point there is a net gain of one original betting unit. The problem is that the certainty of winning requires unbounded available capital.

Fantasy betting

The safest way for the novice to take his first steps into the world of betting is to use his computer to estimate and simulate the forces he will encounter in the real world in which real money changes hands. First generate random uniform integers using Interval Index `I.` to transform each of the numbers in `rnd` into a serial number of one of the intervals defined by the left argument.

```
   wrnd=.(+/\)@[  I. rnd@]   NB. weightd random integers
   >:(to fld)wrnd 10
1 3 3 3 1 4 1 1 2 3
```

thus in 10 reruns the first and third horses each won 4 times, the second and fourth horses won once and the fifth horse not at all. To count frequencies arising in such runs say

```
   +/"1 (i.#fld) =/ (to fld)wrnd 10
2 3 4 0 1
```

which can be consolidated in a verb where the right argument is the number of reruns :

```
   rerun=.dyad :'+/"1 (i.#x) =/ (to x)wrnd y'
   fld rerun 20
10 5 2 2 1
```

that is the favourite won exactly half of the time in the above simulated sequence.

A simulated race with between 5 and 17 runners each of which consists of drawings from a negative exponential distribution with mean 1.25 is given by

```
   sortd=.{~\:
   rne=.[ * ^.@%@rnd@]  NB. random negative exponential
   odds"(0)0.0475>.sortd (%+/)1.25 rne 10
5j1 5j1 6j1 8j1 9j1 10j1 12j1 12j1 14j1 20j1
```

(Note : There is no special reason for using the negative exponential distribution other than that it appears empirically to give lists of odds which look tolerably similar to those actually printed daily in the sporting pages. `0.0475>` is to ensure that no odds are greater than `20j1`.)

It is convenient to head each list with the sequence number of the randomly drawn winner (favourite = 1, etc.).

```
rrace=.monad :0                  NB. random race
r=.odds"(0)t=.0.0475>.sortd (%+/)1.25 rne 5+?13
r=.(>:(+/\ (%+/)t) I. rnd 1),r   NB. join random winner
)
   rrace 10
2 1j1 3j1 12j1 13j1 17j1 18j1 20j1 20j1 20j1 20j1 20j1 20j1
```

A random race card with 3 races is then given by

```
   rrcard=.monad : '>rrace each 5+?y#13'
   rrcard 3
3 2j1 2j1 6j1  6j1  7j1 19j1    0    0    0    0    0    0
1 2j1 3j1 9j1 11j1 14j1 15j1 16j1 17j1 19j1 19j1 19j1 19j1
1 1j1 5j1 5j1  7j1 10j1 18j1 19j1 19j1 19j1 19j1    0    0
```

Betting methods

Various methods can be employed when a bet on a single race is placed, for example the favourite can be backed, or a pin stuck in the race card, or a horse chosen at random but with weights applied based on the quoted odds. These three possibilities are described respectively in

```
method=.dyad : 0
r=.i.0 [ i=.0
while.i<#x do. t=.i{x                  NB. loop through races
select. Y
  case. 1 do. b=.1                     NB. bet on favourite
  case. 2 do. b=.>:?<:#t               NB. stick a pin in race card
  case. 3 do. b=.>:(fro }.t)wrnd 1     NB. random, wts=oddsend.
if.(b={.t)do. r=.r,(<:{.t){%fro }.t    NB. Win
else. r=.r,0 end.                      NB. Lose
i=.i+1 end. R
)
```

The experiments which follow are based on a hypothetical race meeting where between 5 and 17 horses ran in each of 1,000 races, with a simulated 25% overround.

```
   rc=.rrcard 1000
   +/"1>(<rc)method each 1 2 3
752.8 503.3 772
```

gives the total winnings on a unit stake in each race. Thus for each method 1,000 units of were staked, and apart from method 2, the totals in the above run converge towards a value of 800. Repeated reruns with further race cards show consistency in the case of methods 1 and 3 but considerable variability with method 2, which rarely comes even close to 800 – in other words, random selection is likely to be a worst case strategy in the long run! That said, the methods were applied to three real race meetings at Ripon, Carlisle and Newton Abbot with 7, 7 and 6 races respectively with results :

```
Ripon: 9 0 0   Carlisle: 7.375 16 0   Newton Abbot: 11.1 0 3.75
```

showing that even pin-stickers can have their lucky day!

Betting systems

Simulated race cards provide the opportunity for testing out betting systems, that is betting sequences in which stakes change dynamically according to previous results. One such system is due to the 18th-century mathematician d'Alembert. Applying this system, the size of the stake is increased by 1 in the case of a losing bet and decreased by 1 in the event of a winning bet. A zero stake is replaced by the original stake. For example, with an initial bet of 5 and a sole win of 5 on the fourth race out of six, the succession of bets was 5, 6, 7, 8, 7 and 8, a total of 41 for a return of 8×5=40 and an overall loss of 1. The following verb simulates the sequence of stakes :

```
   dalem=.dyad :0                 NB. x is stake, y is returns list
r=.x [ i=.1
while.i<#y do.                    NB. loop through returns
if.(0={:r)do.r=.(}:r),x           NB. if 0 restore initial stake
else.r=.r,({:r)+_1++:0=(<:i){y end.  NB. raise or lower
i=.i+1 end.r
)
   5 dalem 0 0 0 5 0 0
5 6 7 8 7 8
```

Long runs of losers lead to increasingly large stakes developing. Using the 'back the favourite' method on the simulated 1000-race card `rc`, the total returned is

```
   t1=.rc method 1
   +/(*5&dalem) t1
198252.2
```

for total stakes of

```
   +/5 dalem t1
256230
```

198,252/256,230 = 77.4% which is little different from straightforward constant bets. The corresponding figures for methods 2 and 3 are 211,035÷445410 = 47.4% and 267065÷331,278 = 80.6%, indicating again the weakness of 'selecting by pin'. In all cases the figures show how the better runs the risk of a heavy absolute loss using this system when wins are relatively infrequent.

Other systems could be based on patterns of wins and losses for which the primitive verb `E.` is helpful. For example if a constant bet of 5 is made only after observing a 'win-lose' sequence, define

```
    wl=.3 :'0 0,_2}.1 0 E.y~:0'
    wl 1 0 0 0 1 1
0 0 1 0 0 0
```

In this case bets would in the long run be placed only part of the time:

```
    +/wl 0~:t1=.rc method 1
185
    +/(wl t~:0)#t1
131.33
```

131.3÷185 = 71.0% and the corresponding percentages for methods 2 and 3 were 55.5% and 90.0% respectively. The practical message is that neither of the above systems offers the better much hope of advantage in the long run. However, having so much experimental possibility available at home makes things significantly easier to organise than a day at Aintree or Goodwood, and a good deal cheaper too – have a great day in!

FUNCTIONAL CALCULATION

5: Operations

by Neville Holmes (neville.holmes@utas.edu.au)

This article is the sixth in a series expounding the joys of functional calculation. Functional calculation does with operations applied to functions and numbers what numerical calculation does with functions applied to numbers. The functional notation used as the vehicle in this series is provided by a freely available calculation tool called J. This article makes a start to introducing those functional calculation capabilities, in particular the use of certain operators which can be applied to functions and values to produce new functions.

Functional calculation

The description so far has been of numerical calculation, that is, of functions which can be applied to numbers to produce other numbers, though there has been some consideration of structures of characters and boxes. Much is possible using the J notation through such simple numerical calculation, because the notation provides a rich variety of primitive functions, that is, of functions that have symbols like `+` and `<.` and `%:` instead of names given by the user, names like `y` and `foo` and `Herbert`.

What remains to be described is how functional calculation can be built, uniformly and consistently, upon the numerical calculation provided by J.

There are two ways in which functional expressions can be built up – by juxtaposing functions in *trains*, and by applying *operations* to functions and values. Of course the two methods may be combined. Here operations are reviewed while the use of trains is deferred.

In J the definition of functions is exactly the same as the definition of results – it's simply a matter of naming. Naming is done using the `=.` or `=:` copulas, of which the latter is more general in providing a definition which holds globally. Thus the square root of 999 is a numerical result and is named by

```
   sr =: %: 999
```

while the natural log of the difference[1] is a function and is named by

```
   ld =: ^. @ -
```

where the `^.` and the `-` are functions, and the `@` is an operation, as is about to be explained.

Operations

Operations are to functions what functions are to values. Primitive operations are given special symbols, and can be monadic or dyadic, but not both. In this they differ from primitive functions which can be used both monadically or dyadically.

The point about operations is that they are applied to produce functions, whereas functions are applied to produce values, that is, numbers or characters or boxes. As higher level entities, operations apply themselves to their operands more strongly than functions apply themselves to their arguments. Two primitive operations have been briefly considered in previous articles.

The symbol `~` stands for a primitive monadic operation, and its operand (the function to its left) is always applied dyadically. Thus, the argument of the function it produces when used monadically is used both as the left argument and the right argument. Otherwise, the arguments of the function it produces when used dyadically are reversed, or *commuted*, the right argument being used as the left and the left argument being used as the right.

The symbol `/` stands for a primitive monadic operation, and the function it produces when used monadically applies its operand dyadically between all the items of its argument. So `+/` applied monadically to a list of numbers will add them up.

There are two dyadic operations which can be used to combine functions to make new ones, as illustrated in the following diagram.

```
f&g y
```

$$\begin{tikzcd}
f & & \\
& g \arrow[ul] & \\
& & y \arrow[ul]
\end{tikzcd}$$

Monadic After

```
x f&g y
```

$$\begin{tikzcd}
& f & & \\
g \arrow[ur] & & g \arrow[ul] & \\
& x \arrow[ul] & & y \arrow[ul]
\end{tikzcd}$$

Dyadic After

```
f@g y
```

$$\begin{tikzcd}
f & & \\
& g \arrow[ul] & \\
& & y \arrow[ul]
\end{tikzcd}$$

Monadic Of

```
x f@g y
```

$$\begin{tikzcd}
f & & \\
& g \arrow[ul] & \\
x \arrow[ur] & & y \arrow[ul]
\end{tikzcd}$$

Dyadic Of

Syntactically, what holds for primitive functions also holds for composed functions. Where a primitive function can be used so also can a function produced by an operation. A function, primitive or composed, can be used monadically or dyadically, and this is independent of whether any component operation is monadic or dyadic.

Note particularly that an expression like

```
    x f@g y
```

can be keyed in directly for evaluation, but that the expression

```
    x f g y
```

does *not* have the same meaning, even though it might give the same result.

In some of the literature, dyadic operations are called *conjunctions*, while monadic operations are called *adverbs*, by analogy with the conventional names for parts of speech in natural language. This is a dubious analogy, but the names are useful and will be adopted here.

Adverbs

The simpler operations are the adverbs. They only have one operand, to their left; conjunctions have two. Most primitive adverbs are structural, that is, their operand is a function, and the adverb controls how the operand is applied amongst the items of the composed function's argument or arguments. The table lists the adverbs discussed in the following.

`~`	both	swap			
`/`	across	between	`/.`	diagonals	sequester
`\`	prefixes	infixes	`\.`	suffixes	exfixes
`}`	extract	amend	`b.`	basic	
			`f.`	fix	fix

Moving arguments

The simplest primitive adverb has ~ for its symbol. It always uses its operand as a dyadic function, but the function it produces may of course be used monadically or dyadically.

If its result, say `f~`, is used monadically, then the argument is used as both arguments of the operand function. The expression `f~x` is equivalent to `x f x`. For example, `+~x` will double `x`, while `*~x` will square it.

If `f~` is used dyadically, then the arguments are swapped for the operand function. The expression `x f~y` is equivalent to `y f x`. For example, `x-~y` will subtract `x` from `y`, not `y` from `x`, while `x%~y` will divide `x` *into* `y`, not *by* it.

The dyadic use of this adverb is convenient to streamline thought by removing parentheses. Thus (expression) `f x` may be rewritten `x f~` expression.

Inserting functions

The primitive adverb with symbol `/` is structural at a lower level, causing its operand to be inserted between the items of the argument or arguments of the function it produces.

If its result, say `f/`, is used monadically, then it is as though `f` is inserted between the items of the argument of `f/` however many items there might be. In this case `/` is often pronounced *across* or *insert*. For example, `+/x` will add up the items of `x`, `*/x` will multiply them up, and `;/x` will put each in a box.

If `f/` is used dyadically, then it is as though `f` were inserted between the items of the left argument of `f/` and the items of its right argument, each and every one of them at least for scalar functions. In this case `/` is often pronounced *between* or *table*. For example, `x+/y` will, if `x` and `y` are lists, make a table of the sums of the items of `x` and of `y`, while `x*/y` will make a table of their products. Conveniently, `*/~>:i.12` will produce a 12×12 multiplication table, and, using functions other than `*` as operand, other tables may be similarly produced.

The function `f/` is often spoken of as *the f reduction* when used monadically because its effect is normally to reduce the rank of its argument, at least when `f` is a scalar function. The aspect of most interest here is that a list is reduced to a scalar, so that means an empty list like `i.0` or `$9` must reduce to a scalar. That scalar must be the identity value for the reducing function, so that `+/i.0` will yield `0`, while `*/$7` will yield `1`.

More complex ways of inserting functions use symbols that look like the `/` symbol. However, in all cases that follow, the operand is applied monadically by the adverb, unlike the operand of `/` which is applied dyadically.

- Used monadically, `f\` will apply its operand to successive prefixes of its argument. For example, `<\` will show those successive prefixes boxed, and `+/\` will yield progressive sums.
- Used dyadically, `f\` will apply its operand to successive subsequences of its right argument of the size specified by its left argument. When its left argument is negative, the subsequences are consecutive within their argument, if positive their heads are consecutive. Thus `_2+/\y` will yield the sums of distinct pairs of `y`, giving a result with half as many items as `y`, but `2-~/\y` will yield the first differences, that is, it will subtract each

item except the last from its immediately following item giving a result with one item fewer than y.

- The adverb `\.` is just like `\` except the subsequences included by `\` are excluded by `\.` so that `#\i.3` yields `1 2 3` while `#\.i.3` yields `3 2 1`.
- Monadic `/.` applies its operand to diagonals of its argument, while dyadic `/.` applies its operand to subsequences of the right argument selected according to the key given by the left argument.

Other adverbs

The *amend* adverb with symbol `}` behaves in a more complex way. In the first place, its operand may be a function or it may be a value. In the second place, the `}` adverb is closely associated with the function `{`, an unlikely association.

Superficially, monadic `x}` looks like monadic `x&{` when their argument is of rank one. Thus, `4}i.7` yields the same as `4&{i.7` but their behaviour diverges when more complex arguments and operands are used.

Dyadically, after `w=:x z} y` is carried out, `z{w` will yield `x` in simple cases. The basic idea of dyadic amendment is that, using the example just given, the operand `z` specifies which elements of `y` the elements of `x` are to replace.

The *amend* adverb is too complex to explain further here, except that, where the operand is a function, that function is applied to the overall function's argument or arguments to give a result that becomes the effective operand as already described.

A couple of housekeeping adverbs are *basic* and *fix*, spelt `b.` and `f.` respectively. The basic adverb produces a monadic function which, for an argument of `_1` yields a character string showing the obverse of `b.`'s operand, for an argument of `0` a numeric list of the ranks of the operand, and for an argument of `1` a character string showing the identity function of the operand. The obverse is needed for the `~:` and `&.` conjunctions, and can be defined by the `:.` conjunction. The *fix* adverb yields its operand function, but redefined entirely in terms of primitives. All contained definitions are eliminated, so once a function is fixed it can no longer be changed by changing other definitions.

Conjunctions

The more complex operations are the conjunctions, more complex because they have two operands. Some primitive conjunctions are strictly compositional, their operands being functions, and the conjunction controlling how the operands are applied to the composed function’s argument or arguments. Other primitive conjunctions are structural, that is, they have one operation that is a function, and one that is a value which modifies how the other operand, the function, is applied to the conjunction’s argument or arguments.

Here is a table of the simpler conjunctions.

			`;.`	cut	cut	`^:`	power	power
			`!.`	fit	fit	`!:`	foreign	foreign
`"`	rank	rank				`L:`	level	level
`@`	of	of				`@:`	of	of
`&`	after	after	`&.`	dual	dual	&:	after	after

But, before discussing more general primitive conjunctions, it’s useful to review one of the simplest of them, *value bonding*, because it is perhaps the most versatile.

Value bonding

The symbol for bonding is `&`, the ampersand. In value bonding, one of the operands is a function, and one is a value.

For example, used as a monadic function `3&+` will add 3 to its argument, while `%&5` will divide its sole argument by 5. The value operand doesn’t have to be a scalar, nor does it have to be numeric.

On the other hand, used as a dyadic function `a 3&+` will add 3 to its right argument `a` times, while `a %&5` will divide its right argument by 5 `a` times. If `a` is zero, nothing will happen, but if `a` is negative, the inverse function will be carried out `-a` times.

Compositions

Bonding can also be used with two functional operands. For a function composed in this way, the right operand is always used monadically, being applied to each argument when there are two, and the left operand being applied to the result or results from the right operand. Thus `^.&%:y` is the same as `^.%:y` (the *ln* of the square root of `y`), while `x^.&%:y` is the same as `(^.x)%:(^.y)` (in which the second pair of parentheses are included for aesthetic reasons), the *ln* `x` root of the *ln* of `y`.

The other simple function-combining primitive conjunction uses the `@` as its symbol, and applies its left operand to the result of its right operand, which is supplied with whatever argument or arguments the composed function is given.

Thus `^.@%:y` is the same as `^.&%:y`, but `x^.@%:y` is the same as `^.(x%:y)` the *ln* of the `x` root of `y`.

The following diagram shows the nature of these two most common conjunctions.

```
f&g y          x f&g y        f@g y          x f@g y

f                 f           f              f
 ↖               ↗ ↖           ↖              ↖
   g            g   g            g              g
    ↖            ↖    ↖           ↖            ↗  ↖
      y           x     y           y           x    y
```

Monadic After   Dyadic After   Monadic Of   Dyadic Of

The conjunctions which use the `&` and `@` are the most commonly used, but there are several related ones that are of interest.

- The *dual* or *under* conjunction uses the symbol `&.` and applies the obverse (which is usually its inverse) of its right operand to the result of the bond of its operands. This is very useful with a `>` (*unbox*) right operand to allow the contents of boxes to be worked on then put back into boxes.

- For primitive functions the obverse is usually the inverse, but the `:.` conjunction yields its left operand with its obverse defined as its right operand. The obverse is used by the `^:` conjunction described below, as well as by `&.` as just described.

- The symbol `::` stands for the *adverse* conjunction, very like obverse but using its right operand for the replacement upon error of its left operand rather than for its obverse.

- The symbols `..` and `.:` stand for the even and odd conjunctions, though the other uses of the `.` symbol means that usually a blank character must precede these conjunctions to make their meaning unambiguous and plain. In brief, `f ..g` is `-:@(f+f&g)` while `f .:g` is `-:@(f-f&g)`. The names describe the effect when `g=.-` since they then yield the even and odd parts of the function `f`.

Modifying functions

The primitive conjunctions that modify functions typically use the value of their right operand to change the application of their functional left operand to the arguments of the function produced.

Perhaps the most interesting, and at the same time the most complex, of modifying primitive conjunctions is the *rank* conjunction, expressed by the ditto (") symbol. When its left operand is a function, its right operand specifies the rank to be used for the items of the modified function's argument or arguments.

Summary

This article is like a list of ingredients that can be used for combining functions using notation provided by J. At their simplest, these ingredients can be used to define functions that are more complex than the primitive functions but which combine those primitive functions in a variety of ways.

The operations described here, together with those not yet considered, provide the basic means of functional programming. Yet to be considered are trains of various kinds, which will soon be described.

However, the next article in this series will illustrate how operations can be used with functions as a preliminary to treatment of trains.

Postscript

Some explanation of the brevity and content of this article seems needed for new readers of the series and for anyone thinking of using the series for teaching.

The articles were written over a decade ago and were each handed out to students attending a two hour lecture that used the J interpreter's workings displayed on a large screen to show in more detail the meaning of what is gone over in the article handed out at the beginning of the lecture. Any queries by students were answered on the spot by use of the interpreter.

At the end of the lecture students were given an article such as the previous one published here, that is, an article giving many examples of the use of the matter of the lecture. While the exemplary articles were based on using the digits of a year to generate the first hundred integers, the students were required to mimic the examples but using the digits of their student identification number. The code used to mark their work would, they were told, give credit for the variety of expressions and for their brevity.

Notes

1. Properly speaking, the *difference* is the magnitude of the subtraction, but English doesn't have a convenient unambiguous word for the result of a subtraction, so *difference* has its loose meaning here.

P R O F I T

OOStats – A statistics facility for users of Dyalog APL

by Alan Sykes

At the Dyalog Users Conference (Elsinore 2008) I reported my first efforts at providing facilities for statistical computing using Dyalog APL's object-oriented facilities. Since then, the software has expanded and consolidated to such a point that I would like to invite others to use it, suggest suitable extensions to it (and even provide them). This article is therefore a brief introduction to it.

The starting point

From the beginning, I knew that if the software were to be used for real, then it had to cope with missing values. Post retirement, I did some consultancy work, and was invariably given a very messy Excel sheet of data purporting to be a database! Importing this into APL and hitting a column of figures with a simple Mean program had about a 0.1% chance of working! Also, having worked with colleagues into analysing survey data, I knew that as well as system missing values, it was useful to have user-declared missing values for a particular variable (so that analyses of what type of respondents tended to leave a particular database field empty are possible).

So the starting point was the development of a simple database object that allowed the user to do the usual database operations e.g. selecting cases, computing new variables, deleting case *set cetera*. In doing this early work, I soon felt it important to be able to use the graphical user interface (e.g. for declaring variable formats) as well as using the session. (This was fortuitous, as later on, they were incorporated into a full GUI wrap-round that emerged naturally from the object-oriented approach adopted.)

Creating a database from APL

Creating a database from APL should be easy – it is. Using the object code `s_db` in the workspace `oostats`:

```
      db←⎕new s_db (('alan' 'adrian')(alan adrian))
```

The user variables are referred to by names as listed in the public field:

```
      db.UserNames
 alan  adrian
```

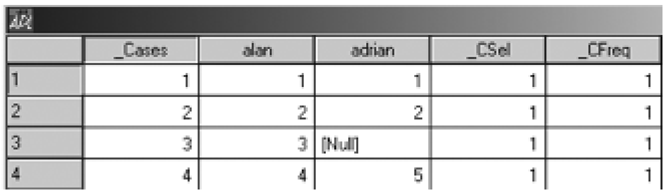
In addition, however, there are three variables that keep a track of the case number, whether that case is selected, and the case frequency:

```
      db._Cases
1 2 3 4
      db._CSel
1 1 1 1
      db._CFreq
1 1 1 1
```

(Occasionally, it is helpful to be able to declare a case frequency – for example if analysing a contingency table given from external sources.)

To look at the database:

```
      db.View
```



	_Cases	alan	adrian	_CSel	_CFreq
1	1	1	1	1	1
2	2	2	2	1	1
3	3	3	[Null]	1	1
4	4	4	5	1	1

Figure 1

```
 System missing values are kept in a field: db_SM
      (⎕null)(⊂'')(⊂⍬)('')∊db._SM
1 1 1 1
```

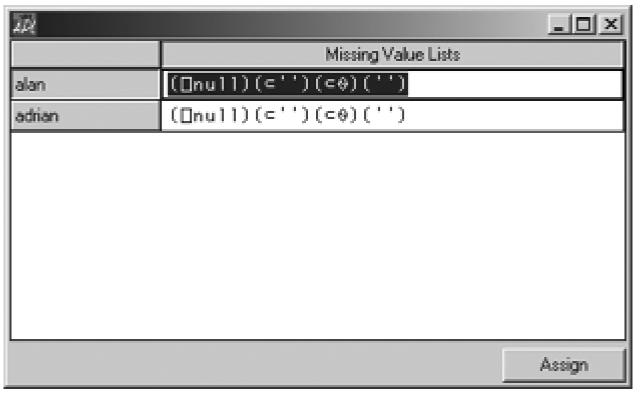
and the collection of missing values for each of the user variables is contained in

```
      db.MissVals
  (⎕null)(⊂'')(⊂⍬)('')  (⎕null)(⊂'')(⊂⍬)('')
```

Each of these lists is a string (to make it easier to see just what the missing values are) and may be added to:

```
      db.MissVals[1],←⊂'(2)'
      db.MissVals
 (⎕null)(⊂'')(⊂⍬)('')(2)  (⎕null)(⊂'')(⊂⍬)('')
```

Alternatively, the method `db.GetMissVals` provides a grid object for entering further values:



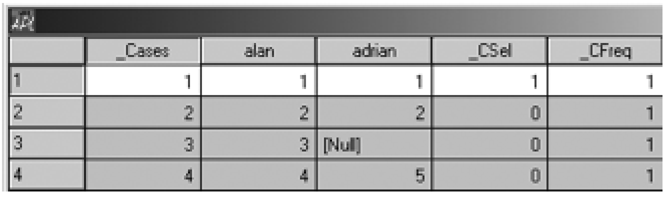
	Missing Value Lists
alan	(⎕null)(⊂'')(⊂⍬)('')
adrian	(⎕null)(⊂'')(⊂⍬)('')

Figure 2

After such an allocation, any statistical method using the variable `alan` would filter out values equal to 2.

Selecting cases is programmed as a method in `s_db`, and is straightforward:

```
      db.SelectCases '(alan<4)^adrian<5'
Cases selected by (alan<4)^adrian<5
3 cases not selected
      db.View
```



	_Cases	alan	adrian	_CSel	_CFreq
1	1	1	1	1	1
2	2	2	2	0	1
3	3	3	[Null]	0	1
4	4	4	5	0	1

Figure 3

In the grid view of the database, cases not selected are in grey – note that case 3 has not been selected because I take the view that a null value cannot be included in the comparison.

Statistical Methods

As well as the above (and other) database methods, the object code `s_db` contains a number of statistical methods:

```
       db.StatsMethods
UniqueFrequency Unistats Regress TwoSampleMeans CrossTabs
Multistats MatchedPairs OneWayAnova Scatterplot Table Boxplot
TimeSeries OneWayManova
```

With one exception (the `Unistats` method) each statistical method creates a sub-object `db.s` which has its own fields and methods. For our first example, consider `UniqueFrequency`, useful when investigating a database for the first time – it lists unique values of a variable and their frequency (missing values are included here):

```
       db.UniqueFrequency 'adrian'
 Sub-object s created from all cases
       )cs db
 #.[s_db]
      s.UniqueValues
 1 2 5  [Null]
      s.Frequencies
 1 1 1 1
      #.Tab s.FrequencyTable
  ValueLabels  Values  Frequency  Percentage
               1               1          25
               2               1          25
               5               1          25
               [Null]          1          25
```

(With a view to printing out tables later, a table is returned as a vector of column headings and then the body of the table – `#.Tab` simply glues them together adjusting lengths as necessary.)

The Unistats sub-object

Perhaps the most frequently used statistical method in `s_db` is the `Unistats` object which allows you to calculate means and standard deviations etc for a single variable. Using this from the session, I took the view that one might want to do this for more than one variable, so I decided to create an object at the root level called by the name of the variable.

```
       db.SelectCases 1
 (re-instates all cases)
       db.Unistats 'adrian'
 Created object #.adrian.?
 Currently, # cases excluded = 1
```

(In creating this object, any value that is in the list of missing values for the variable is filtered out. The ability to do this automatically when creating a statistical object is really important. For example, when fitting different competing regression models, cases will be included or excluded as each new model is specified.)

We can now type:

```
      adrian.Mean
2.66667
      adrian.StDev
2.08167
```

The choice of options for the `Unistats` method reflects my own personal outlook on the process of statistical data analysis – in particular the important role that graphics plays in understanding what is going on and therefore what analysis is (or is not) relevant. So, for example, with `Unistats` – there are three graphs – a histogram (`Hist`), a `Boxplot` (useful for identifying cases that are outliers) and a `Rankitplot` (a visual check on whether or not the data is Normally distributed). Other options are easily added.

Finding out about the options available

With any statistical object, it is useful to document what options are available, and also to provide a *Script* (a nested matrix) for documenting output. This is driven by the function `#.Explore`. For example, the statistical method `UniqueFrequency` has a Script matrix

```
      #.Ed.freq
 1 1  Heading         1   2  Heading
 1 0  UniqueValues    1   2  'Unique Values'
                      1   3  UniqueValues
 1 0  Frequencies     1   2  'Frequencies'
                      1   3  Frequencies
 1 1  FrequencyTable  1   2  'Frequency Table of all Unique Values'
                      1   4  FrequencyTable
```

listing options (plus information on left and right arguments) with the last column specifying executable commands for output if that option is selected. (The matrix itself may be constructed through a specially designed GUI nested matrix editor `#.Ed.Edit`.)

The object itself accesses this information through a field `s.Options`:

```
      s.Options
  Left arg  Option         Right arg
            Heading
            UniqueValues
            Frequencies
            FrequencyTable
```

To see this working more effectively,

```
      Open 'c:\oostats\student.adb'
 Object 'db' has been created using s_db from file
 c:\oostats\student.adb
```

(A database is saved in an APL component file together with its attributes.)

```
      db.UserNames
  sex  height  weight  age  left  react  sort
      db.Unistats 'weight'
 Created object #.weight.?
 Currently, # cases excluded = 0
      weight.Options
  Left arg                  Option               Right arg
                            Heading
                            Sum
                            Mean
                            StDev
                            StError
                            LHinge
                            Median
                            Uhinge
                            Table of Statistics
                            Outliers
                            ExtremeOutliers
                            Ttest                hypval←0
                            NonParTest           hypval←0
                            Percentiles          pcts←25 50 75
                            FreqTable            start,width←
                            ConfInt              conflev%←95
  Normal,Exponential,Gamma… Hist                 start,width←
                            Boxplot
                            Rankitplot
```

Right arguments (numeric) are indicated through a text vector, thus the `Ttest` option has a right argument which specifies the hypothesis value required. Left arguments, where they exist, are a list of names indicating categorical options – thus for the `Histogram` method, a left argument of `'Normal'` would add a normal-density overlay to the histogram.

Here are some examples of the options:

```
      weight.Min
33
      weight.Max
96
      weight.Hist 30 4
```

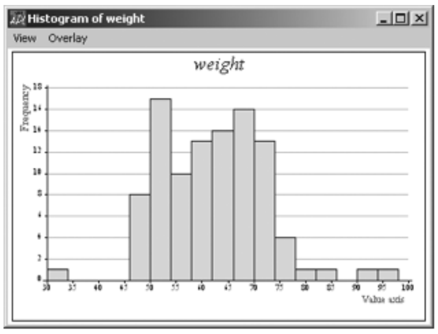
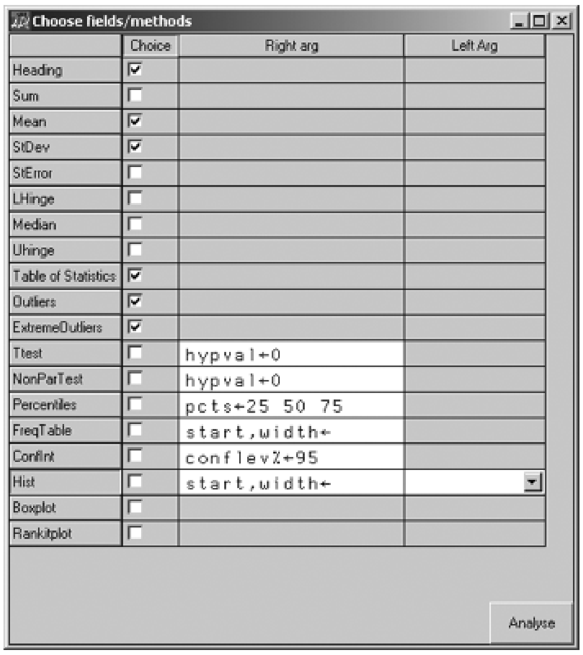


Figure 4

Note the menu item `View` which activates the Causeway viewer, and `Overlay` which gives you a choice of fitting to the data a Normal, Exponential, Log-Normal, Gamma distribution or a smoothed version of the histogram (the left argument options to `Hist`).

If you are unsure of which options to use, then you can use `#.Explore` with right argument equal to the object to be explored. This allows the user to tick appropriate options and obtain suitably annotated output:

```
      #.Explore weight
```



	Choice	Right arg	Left Arg
Heading	✓		
Sum			
Mean	✓		
StDev	✓		
StError			
LHinge			
Median			
Uhinge			
Table of Statistics	✓		
Outliers	✓		
ExtremeOutliers	✓		
Ttest		hypval←0	
NonParTest		hypval←0	
Percentiles		pcts←25 50 75	
FreqTable		start,width←	
Confint		conflev%←95	
Hist		start,width←	
Boxplot			
Rankitplot			

Figure 6

```
Univariate statistics for weight
 Statistic    Value
 #-cases     100
 Sum        6188
 Mean         61.88
 StDev         9.98107

Outliers

There are 2 outliers

 Case  Value
   77    92
   95    96

Extreme Outliers

There are 0 extreme outliers
```

If we pursue the information on weight of students further, we can recognise that there are male and female students in the same data set, so this histogram is a mixture of two distributions (one for each sex). To investigate how they differ, we need either a boxplot (see later), or two histograms on one axis (not advisable and so not provided) or two smoothed histograms. Both options are available if we use the statistical method `TwoSampleMeans`:

```
      db.TwoSampleMeans 'weight' 'sex=1' 'sex=2'
Sub-Object s created using s_twosamt
      db.s.FreqDensities 4
```

(The parameter is a smoothing parameter – think of it as a class-width for a histogram.)

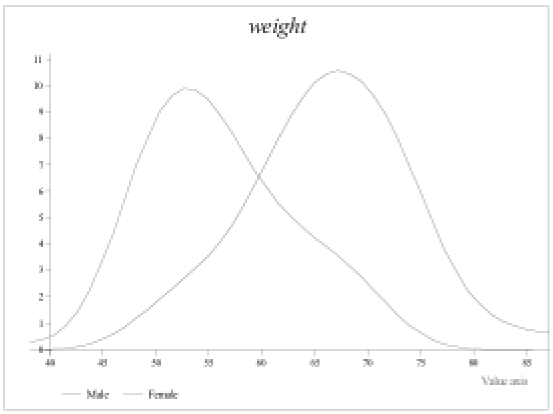


Figure 6

Note that the labels in the key are produced from the database, which has a field `db.ValueLabels` – a vector of matrices, one for each user variable:

```
      db.ValueLabels
 1  Male
 2  Female
```

(n.b. there is only one variable here with value labels)

The resulting graph gives a clear picture of how the distributions of male and female weights differ – a formal test of the equality of means may be performed (either assuming approximate normality or using a non-parametric test):

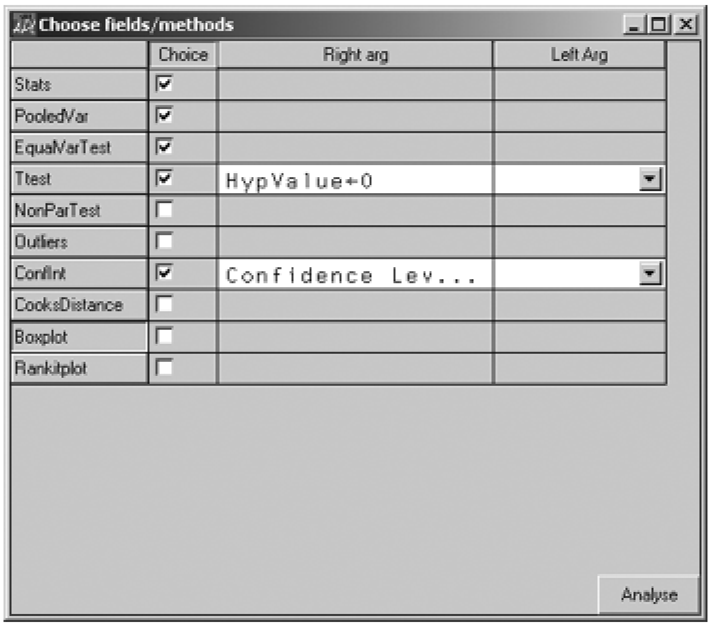
```
      Tab db.s.EqualVarTest
 df1  df2  F-statistic   p-value
  53   45      1.21619  0.251715
```

```
which tells us that we can assume equal variances (as suspected
from the two densities above)
```

```
      Tab db.s.Ttest 0
 t-Statistic  df      p-value
     7.10045  98  1.99094E¯10
```

```
If you prefer to use the #.Explore method, then it is a little
easier to see what is going on
```

```
      #.Explore db.s
```



Choose fields/methods

	Choice	Right arg	Left Arg
Stats	✓		
PooledVar	✓		
EqualVarTest	✓		
Ttest	✓	HypValue←0	
NonParTest	☐		
Outliers	☐		
ConfInt	✓	Confidence Lev...	
CooksDistance	☐		
Boxplot	☐		
Rankitplot	☐		

Analyse

Figure 7

The chosen options are then performed and reported back with annotations:

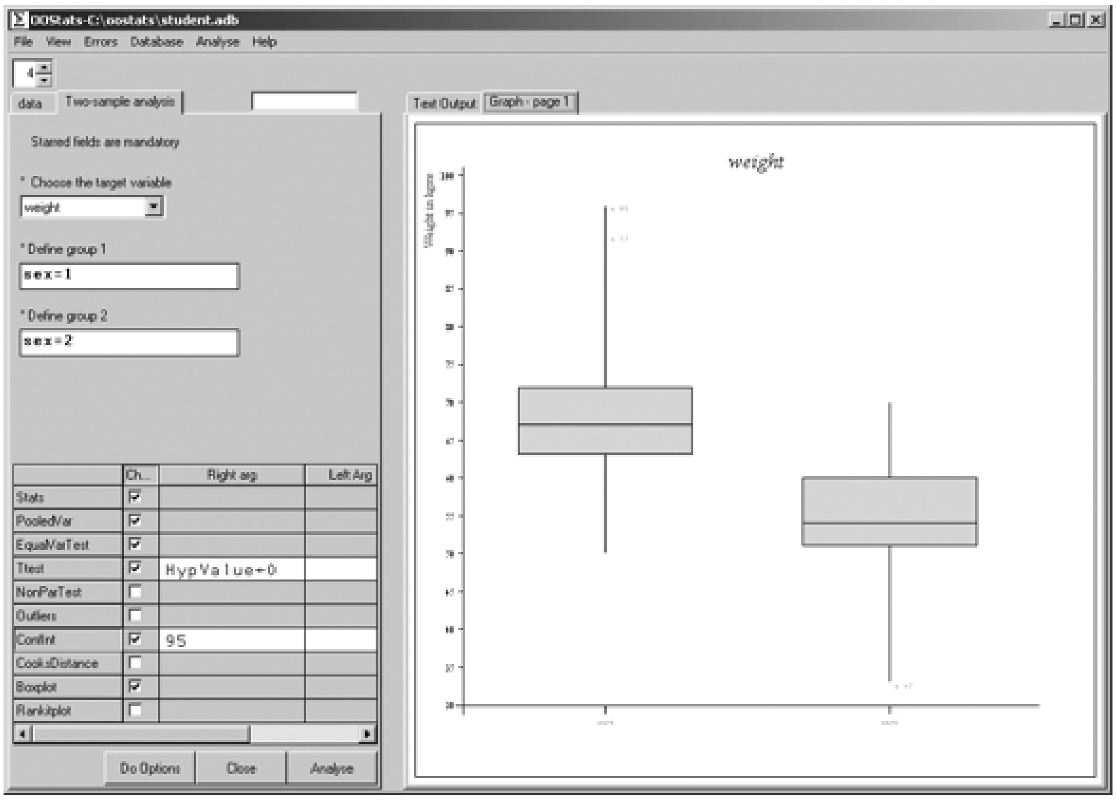
```
Two-sample analysis for variable weight

Group 1 is defined by the statement sex=1
Group 2 is defined by the statement sex=2

Sample Statistics

          group 1  group 2
 Means     67.22    55.61
 St Devs    8.835    7.265
 #-cases   54       46

Pooled Variance Estimate

 Estimate  Degrees of Freedom
    66.45                  99

Test of Equality of Variances

 df1  df2  F-statistic  p-value
  53   45        1.216   0.2517

t-Test of Hypothesis that the means differ by 0

The results following assume equal variances

 t-Statistic  df    p-value
        7.1   98  1.991E¯10

95% Confidence Interval for Difference of Two Means

The results following assume equal variances

 Lower value  Upper Value
       7.89        15.34
```

The Graphic User Interface

Whilst driving a statistical analysis from the keyboard is a familiar environment for statisticians, a GUI interface is also desirable. This is provided by the object code `s_guidb`, which inherits the properties and methods of `s_db`. Because of the inheritance, and because all the database facilities already have a GUI interface (e.g. `db.GetMissVals`) it is straightforward to incorporate them into a menu-driven system.

For the statistical methods, forms are necessary to declare the appropriate variables to spawn the statistical object – once the object has been created, the grid object used in `#.Explore`, provides the user choice for the statistical options required from that object, and the output from Explore gives the output required for an RTF-viewer. This is illustrated by using the male and female weights example again.

Having selected from the Analyse menu, the option Two-sample analysis, we can select the target variable, specify the two groups, and create the object using the Analyse button. The default options can then be executed by pressing Do Options on the tabbed subform. The boxplot is generated on the right-hand tabbed subform as seen below.



Starred fields are mandatory

* Choose the target variable: weight

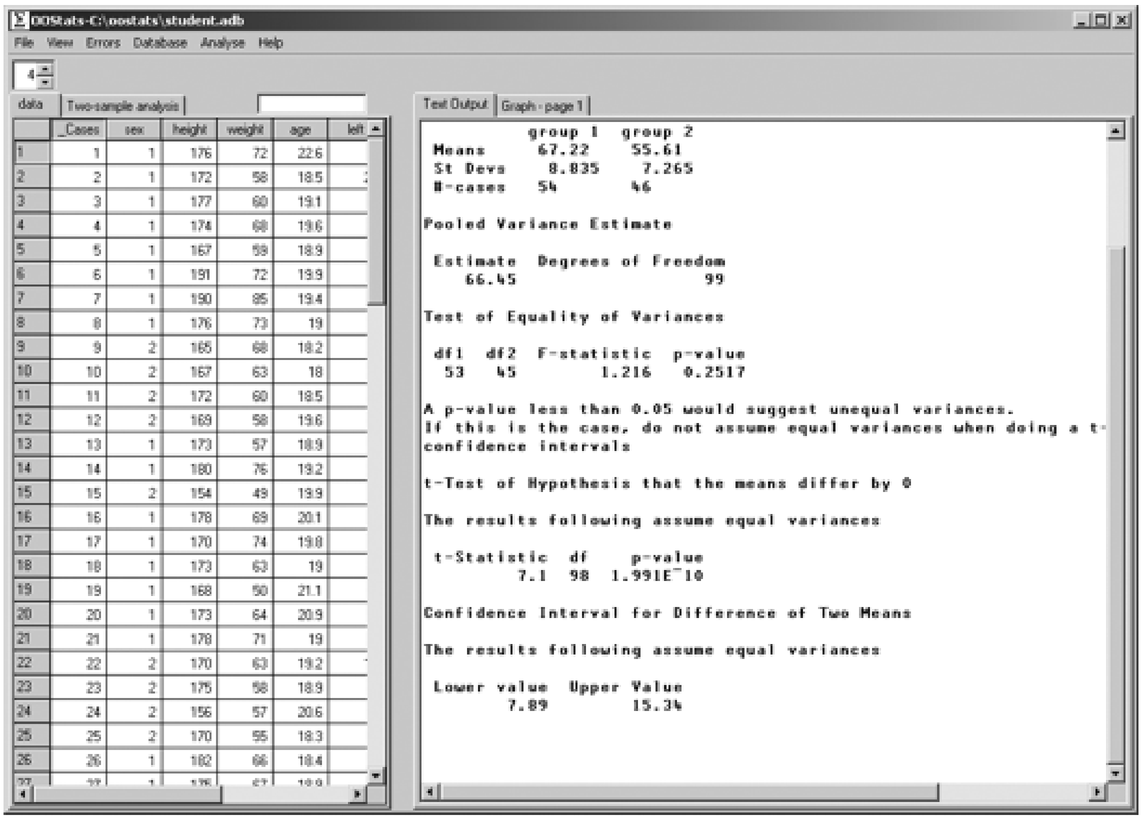
* Define group 1: sex=1

* Define group 2: sex=2

	Ch...	Right arg	Left Arg
Stats	✓		
PooledVar	✓		
EqualVarTest	✓		
Ttest	✓	HypValue←0	
NonParTest	☐		
Outliers	☐		
ConfInt	✓	95	
CooksDistance	☐		
Boxplot	✓		
Rankitplot	☐		

Figure 8

The hidden tabbed sub-forms reveal the database grid object and the text output:


OOStats-C:\oostats\student.adb
File View Errors Database Analyse Help
4
data
Two-sample analysis
Text Output
Graph - page 1

	_Cases	sex	height	weight	age	left
1	1	1	176	72	22.6	
2	2	1	172	58	18.5	
3	3	1	177	60	19.1	
4	4	1	174	68	19.6	
5	5	1	167	59	18.9	
6	6	1	191	72	19.9	
7	7	1	190	85	19.4	
8	8	1	176	73	19	
9	9	2	165	68	18.2	
10	10	2	167	63	18	
11	11	2	172	60	18.5	
12	12	2	169	58	19.6	
13	13	1	173	57	18.9	
14	14	1	180	76	19.2	
15	15	2	154	49	19.9	
16	16	1	178	69	20.1	
17	17	1	170	74	19.8	
18	18	1	173	63	19	
19	19	1	168	50	21.1	
20	20	1	173	64	20.9	
21	21	1	178	71	19	
22	22	2	170	63	19.2	
23	23	2	175	58	18.9	
24	24	2	156	57	20.6	
25	25	2	170	55	18.3	
26	26	1	182	66	18.4	

```
           group 1  group 2
 Means      67.22    55.61
 St Devs     8.835    7.265
 N-cases    54       46

Pooled Variance Estimate

 Estimate  Degrees of Freedom
    66.45                  99

Test of Equality of Variances

 df1  df2  F-statistic  p-value
  53   45        1.216   0.2517

A p-value less than 0.05 would suggest unequal variances.
If this is the case, do not assume equal variances when doing a t-
confidence intervals

t-Test of Hypothesis that the means differ by 0

The results following assume equal variances

 t-Statistic  df    p-value
         7.1  98  1.991E¯10

Confidence Interval for Difference of Two Means

The results following assume equal variances

 Lower value  Upper Value
        7.89         15.34
```

Figure 9

Users wishing to extract output into, say Microsoft Word, can

1. copy and paste from the RTF Viewer,
2. paste any of the graphs produced (some objects may have up to three different graphs) from the Causeway Viewer available on the `View` menu and
3. print a Newleaf report of all (or selected parts) of the activity in a session.

The `Help` menu provides a set of Help files (standard compiled html) produced using Adrian Smith's documentation software. Other database features not mentioned include formatting the variables (including showing dates and value labels), ordering cases, ordering the variables, using colour in the grid to indicate the spectrum of small to large values, and the ability to edit the grid if required.

The scope of OOStats

Currently, OOStats is for APLers using Dyalog 12.1. Some options for further development are obvious:

- the addition of further statistical methods to `s_db`
- the addition of further options to any of the existing statistical methods

- packaging it up to provide a stand-alone product (it would be necessary to provide some cover functions for use in e.g. Case Selection or Computing a new variable)
- Regularising the extended output by the creation of a dictionary thus allowing output in different languages.

(Readers may wish to extend the list!)

The table below lists the statistical scope to date – note that all the analytic power of ASLGREG is available (facilitating some quite advanced analyses on multi-way contingency tables, logistic regression etc.) and I would hope that there is much here that statistical APLers could use.

Statistical Method	**Options**		
Boxplot	*Facilitates boxplots for one or more variables including classifying variables* (This is a cover-method to interface with boxplots provided on other sub-objects)		
CrossTabs	Observed	Expected	ViewFullTable
Analysis of two-way	Table	StandResid	BarchartVar1ByVar2
frequency tables	ProportionsVar1ByVar2	ChiSquareTest	BarchartVar2ByVar1
	ProportionsVar2ByVar1	FullTable	TowerChart
Table	Table		
Provides a table of	Boxplot		
univariate statistics			
within groups specified			
by one or two variables			
MatchedPairs	*Creates a Unistats object on the difference of two variables – see below*		
MultiStats	#.Cases	75%ile	ExtremeOutliers
Creates an object	Mean	95%ile	TsquareTest
for the univariate	Std Dev	UnivariateStats	TestHypEqualMeans
or multivariate	5%ile	MultivariateStats	CorrelationMatrix
analysis of a group	25%ile	CovarianceTable	CorrelationTable
of variables	50%ile	Outliers	
OneWayAnova	MeansTable	GroupContrasts	FreqDensities
Analysis of one	EqualVarTest	CooksDistance	Boxplot
variable split into	AnovaTable	Outliers	Rankitplot
two or more groups	Ftest		
OneWayManova	MeanVector	BGCovarianceMatrix	ParallelProfileTest
As above, but for a	GroupMeanMatrix	HonogeneityTest	GroupContrasts
group of correlated	GroupMeansTable	WilksLambda	MeansPlot

variables	WGCovarianceMatrix	HotellingsTsquare	CanonicalPlot
Regress	CurrentModel	CorrelationMatrix	Leverage
Regression and	AnovaTable	Outliers	CooksDistance
Generalized Linear	Ftest	Diagnostics	Stepwise
Modelling	DevianceTable	FittedValues	Parityplot
	EstimateTable	StandardisedResiduals	Fitplot
	CovarianceMatrix	TResiduals	RankitPlot
Scatterplot	*Allows the building of any regression or generalized linear model involving a y-variable, one regressor variable, and one factor variable showing a scatterplot with fitted model*		
TimeSeries	Acf	ARModel	ARMAModel
Fits auto-regression or moving average time-series models	Pacf	MAModel	Plot
TwoSampleMeans	Stats	NonParTest	FreqDensities
	PooledVar	Outliers	Boxplot
	EqualVarTest	ConfInt	Rankitplot
	Ttest	CooksDistance	
UniqueFrequency	UniqueValues	Frequencies	FrequencyTable'
Frequencies of unique values			
Unistats	Sum	Uhinge	Pctile
Statistics for one	Mean	Max	FreqTable
variable	StDev	Outliers	ConfInt
	StError	ExtremeOutliers	Hist
	Min	BoxCoxLL	Boxplot
	Lhinge	Ttest	Rankitplot
	Median	NonParTest	

Acknowledgements

I am indebted to Morten Kromberg and Gitte Christensen of Dyalog Ltd for encouraging the work on this project and for the opportunity to present a preliminary version at the 2008 Users Conference. As with nearly all my endeavours in APL, I have benefitted enormously from the large APL toolkit of Adrian Smith. Specifically here, is the use of Rainpro, Newleaf and the Documentation software, but not Causeway – the GUI coding, however inept, is my own!

Attachments

1. Dyalog APL workspace, and examples of datasets:
   `http://archive.vector.org.uk/content/published/sykes/oostats.zip`

2. Compiled HTML Help file:
   `http://archive.vector.org.uk/content/published/sykes/oostats.chm`

Odd-order magic squares expressed in J

by John C. McInturff

This note illustrates an array-oriented approach to solving odd-ordered magic squares. Part 1 illustrates a computer-sensible solution expressed in J. The resulting solution is then subjected to eight symmetrical transformations and each tested for 'magic properties'. Part 2 describes the underlying two-step rule for the solution, and illustrates how this rule can be applied to the entire magic square (matrix), and carried out graphically without a computer. Each graphical step is made computer-sensible and is executed.

Part 1

The following verb, `MS`, is an array-oriented solution to an odd-ordered Magic Square, expressed in J. This expression is intended to minimise keystrokes, not maximise the understanding of the thought behind it. The latter objective is addressed in Part 2.

```
   MS=. 3 : ' (1|. N) P |: (N=. ,.i.y) (P=. |."1) (<.-:y) |. >:i.2#
 y'
```

Shown below are four examples, for `n=. 3 5 7 9`. The verb, *each*, is: `ea=. &.>`

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

The sum of each row, column, right diagonal, and left diagonal, is required to be equal to the value known as the *magic value* produced by the verb `val=. [:-:]*1+*:`.

The magic value for *each* magic square above is therefore:

```
   val ea 3 5 7 9
+-------------+
¦15¦65¦175¦369¦
+-------------+
```

The question is: do the above matrices satisfy the ‘magic requirement’? The answer requires n calculations where n is equal to the magic value plus the two diagonals plus double the number of sides for each matrix. For matrices of order 3, 5, 7, 9 and 41, n would be:

```
   ]n=. (1+2++:) ea 3 5 7 9 41
+-------------+
¦9¦13¦17¦21¦85¦
+-------------+
```

The objective now is to see if the four matrices meet the above conditions. The sum of each row, column, left diagonal, and right diagonal is given by the following verbs:

```
   row=. +/"1
   col=. +/"2
   d1=.[: +/ (<0 1) |: ]
   d2=. [: +/ (<0 1) |: |.
   v=. [: val #
```

For brevity, the following verb `f` combines the above 5 verbs and illustrates an example of its use for `n=. 5`

```
   f=. (v;' ';row;col;d1;d2)
   f (MS 5)
+---------------------------------------+
¦65¦ ¦65 65 65 65 65¦65 65 65 65 65¦65¦65¦
+---------------------------------------+
```

It is seen that the order-5 matrix took thirteen calculations and met all conditions. An order-41 matrix would require 85 conditions. The following verb `Test`, when applied to the matrix order, takes all of the above requirements into account and returns a 1 if all conditions are met; e.g.,

```
   Test=. 3 : 0
   n=. val y
   q=. MS y
   *./n=(,>(row q)),(col q)),(d1 q),(d2 q)
   )
```

This will now be illustrated for all odd matrices from order 3 through 41.

```
   odd=. (1+2*i.21)
   n=.(3++:) odd
   test=. Test ea odd

   >< ea odd,n,:(>test)
+-------------------------------------------------------------+
¦1¦3¦5 ¦7 ¦9 ¦11¦13¦15¦17¦19¦21¦23¦25¦27¦29¦31¦33¦35¦37¦39¦41¦
+-+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--¦
¦5¦9¦13¦17¦21¦25¦29¦33¦37¦41¦45¦49¦53¦57¦61¦65¦69¦73¦77¦81¦85¦
+-+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--¦
¦1¦1¦1 ¦1 ¦1 ¦1 ¦1 ¦1 ¦1 ¦1 ¦1 ¦1 ¦1 ¦1 ¦1 ¦1 ¦1 ¦1 ¦1 ¦1 ¦1 ¦
+-------------------------------------------------------------+
```

The value of a magic square is unaffected by the following eight transformations. These are the identity transformation `t0`, plus three clockwise rotations and their four respective reflections. There are, therefore, eight magic squares associated with the verb `MS`.

These eight transformations are expressed by the following eight verbs and are illustrated below:

```
   t0=: ]
   t1=: t6@t7
   t2=: t4@t6
   t3=: |.@|:
   t4=: |.@]
   t5=: t2@t7
   t6=: |."_1@]
   t7=: |:@]
```

For brevity, these eight transformations are combined into the single verb, `t`, and are more clearly illustrated below for the rectangular 3×2 matrix, `a`.

```
   t=. (t0;t1;t2;t3;t4;t5;t6;t7)
   ]a=. >:i.3 2
1 2
3 4
5 6
   ]T=. t a
+---------------------------------------+
¦1 2¦5 3 1¦6 5¦2 4 6¦5 6¦6 4 2¦2 1¦1 3 5¦
¦3 4¦6 4 2¦4 3¦1 3 5¦3 4¦5 3 1¦4 3¦2 4 6¦
¦5 6¦     ¦2 1¦     ¦1 2¦     ¦6 5¦     ¦
+---------------------------------------+
```

The eight magic squares are therefore:

```
   t (MS 3)
+-----------------------------------------------+
¦8 1 6¦4 3 8¦2 9 4¦6 7 2¦4 9 2¦2 7 6¦6 1 8¦8 3 4¦
¦3 5 7¦9 5 1¦7 5 3¦1 5 9¦3 5 7¦9 5 1¦7 5 3¦1 5 9¦
¦4 9 2¦2 7 6¦6 1 8¦8 3 4¦8 1 6¦4 3 8¦2 9 4¦6 7 2¦
+-----------------------------------------------+
```

Part 2

The third magic square of the eight above is the 650-BCE Lo Shu magic square[1], often credited as being the first recorded magic square. (Somehow it got on the back of a turtle!)

The French diplomat Simon de la Loubère 1642-1749[2] published the following “well known” rule for solving odd-ordered magic squares.

1. Initialise a square grid (matrix) by placing the integer 1 in the center column of the first row.
2. Place the next number, 2, in the square diagonally up and to the right.
    1. If filled, move vertically down one square,
    2. If ‘off the square’, wrap around (odometer-wise) to the last row, or first column, respectively.
3. Continue with the next number 3 etc. (repeating the above rule if necessary) until the square is filled.

One can start with any number other than 1 and follow the above rule to derive other magic squares belonging to the group of eight mentioned in Part 1.

Although the above rule involves a single number at a time and an iterative process, it can be carried out extremely rapidly by hand. Furthermore, it is the basis for the thought process that takes place when one applies the same principle to the matrix as a whole and follows the same two-step rule.

1. The first step is to initialise the matrix such that the integer 1 will always appear in the centre column of an outside edge. This is assured by verb `f2` in the Appendix.
2. The second step is to move the entire matrix diagonally up and to the right. The term *move* is a vector instruction that successively shifts each row of the matrix, as illustrated in the Appendix.

Two important comments must be made:

- For the array-oriented language used, it may be more desirable to 'translate' the matrix and move equivalently, left and up as was done in the verb `MS`.
- *Left* and *up* can be thought of as two degrees of freedom. If one has only one degree of freedom, it can be equivalently accomplished by an anti-clockwise translation and a left shift as was done in the verb `MS`.

The array approach using the above method can be carried out by hand as well as executed by a computer as described in detail in the Appendix.

References

1. Lo Shu Square
   http://en.wikipedia.org/wiki/Lo_Shu_Square

2. Simon de la Loubère
   http://en.wikipedia.org/wiki/Simon_de_la_Loub%C3%A8re

Appendix

GRAPHICAL

0. Populate Matrix

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

1. Initialize Matrix

11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
1	2	3	4	5
6	7	8	9	10

2a. LEFT shift

11	12	13	14	15
17	18	19	20	16
23	24	25	21	22
4	5	1	2	3
10	6	7	8	9

2b. UP (Transpose)

11	17	23	4	10
12	18	24	5	6
13	19	25	1	7
14	20	21	2	8
15	16	22	3	9

2b: UP (LEFT shift)

17	23	4	10	11
24	5	6	12	18
1	7	13	19	25
8	14	20	21	2
15	16	22	3	9

COMPUTER

```
  y=. 5                  ]a=. f1 y
  f1=. [:>:[:i.2#]    1  2  3  4  5
                      6  7  8  9 10
                     11 12 13 14 15
                     16 17 18 19 20
                     21 22 23 24 25
```

```
  (f2 y);a                    ]b=. (f2 y) |. a
┌─┬──────────────┐        11 12 13 14 15
│2│ 1  2  3  4  5│        16 17 18 19 20
│ │ 6  7  8  9 10│        21 22 23 24 25
│ │11 12 13 14 15│         1  2  3  4  5
│ │16 17 18 19 20│         6  7  8  9 10
│ │21 22 23 24 25│
└─┴──────────────┘
```

```
  n=. [:,.i.
  N=. n y
  N;' ';b
  P=.|."1                          ]c=. N P b
┌─┬─┬──────────────┐        11 12 13 14 15
│0│ │11 12 13 14 15│        17 18 19 20 16
│1│ │16 17 18 19 20│        23 24 25 21 22
│2│ │21 22 23 24 25│         4  5  1  2  3
│3│ │ 1  2  3  4  5│        10  6  7  8  9
│4│ │ 6  7  8  9 10│
└─┴─┴──────────────┘
```

```
                                 ]d=. |: c
                              11 17 23 4 10
                              12 18 24 5  6
                              13 19 25 1  7
                              14 20 21 2  8
                              15 16 22 3  9
```

```
  k1=. monad : '1|. ,.i.# y'
  K=. k1 N
  K;' ';d                         ]e=. K P d
┌─┬─┬─────────────┐         17 23  4 10 11
│1│ │11 17 23 4 10│         24  5  6 12 18
│2│ │12 18 24 5  6│          1  7 13 19 25
│3│ │13 19 25 1  7│          8 14 20 21  2
│4│ │14 20 21 2  8│         15 16 22  3  9
│0│ │15 16 22 3  9│
└─┴─┴─────────────┘
```

What is it about infinity?

Sylvia Camacho in conversation with Graham Parkhouse

A conversation about some paradoxes of infinity, helped by J notation.

Anthony and I contrive to live among a chaos of possessions acquired during our own seventy-odd years and a goodly proportion added by inheritance. We have walls of shelves but any other horizontal surface attracts books, which are never discarded. Indeed there are occasional duplicates bought anew to replace a valued volume gone missing without trace. We are often taken by surprise by new arrivals which, as the bailiff never comes knocking, we presumably ordered and paid for. However I was slightly taken aback to find three books about prime numbers all published since 2000 and all telling the same story. Readers should appreciate that neither Anthony nor I have more than School Certificate mathematics. So what was so compelling about these three that I began to explore one of them in detail, using J to try to connect to its mathematical arguments?

All three of our books about primes were published inside two years, as a response to the centenary of David Hilbert's 1900 Address to the International Congress of Mathematicians, challenging them with 10 mathematical problems still looking for resolution at the start of the 20th century. For our three authors it was noteworthy that number 8 among these, the Riemann Hypothesis, was still unproven after yet another century, at the start of the 21st. All three books recount the history of attempts to prove the hypothesis, delivered by Bernard Riemann to the Berlin Academy in 1859 and entitled *On the Number of Prime Numbers Less Than a Given Quantity*.

This would have been a matter of only passing interest for me but for the pivotal story, common to all three, of the chance encounter between numerical analyst Hugh Montgomery and mathematician and physicist Freeman Dyson, at Princeton in 1972. This revealed a possible connection between Riemann's attempts to characterise the distribution of prime numbers using the tools of calculus and the much later use of matrices in quantum theory; although, of course, each investigation was in pursuit of wholly different ends.

I have read and tried to summarise histories of calculus that describe how it progressed from a tool justified by little more than Isaac Newton's intuition, to a fully axiomatic system for defining the concept of a continuum. The assumption of a spacial and temporal continuum was a natural consequence of the astronomical speculations of the 17th century, but it was the depth of penetration of calculus into almost all pre-1925 physics that made its failure to mathematize quantum effects so traumatic that it led to a schism within the physics community. One would have supposed that there is nothing more quantal than the succession of natural numbers and yet Riemann's proposition was that the techniques of calculus could be used to predict the distribution of primes across the infinity of whole numbers. So could that 1972 encounter at Princeton presage a healing of the schism?

To date I have only come to terms with about half of the book I chose to study. It is by John Derbyshire and entitled *Prime Obsession*. When I am wholly lost I turn to my friend Graham Parkhouse, who has put my feet back on the right path several times as I struggle to express Derbyshire's exposition in J. He encouraged me to come to terms with Equation Editor so I am now able to show here what all the fuss was about. Riemann's Hypothesis is:

> All non-trivial zeros of the zeta function have real part one-half.

For a non-mathematician this is obscure, but at least I know what a Greek zeta looks like (ζ) and I do have a mathematical dictionary which says:

> The zeta function of complex numbers z = x + iy is defined for x > 1 by the series…

$$\zeta(z) = \sum_{n=1}^{\infty} n^{-z} = \sum_{n=1}^{\infty} e^{-z \log n}$$

Even I can see that this is an infinite sum and it becomes apparent in the course of the book that, given mathematical ingenuity, it can be evaluated for any real number over the entire complex plane, with the sole exception of integer 1. So it is natural that the characteristics of infinite series are a key part of Derbyshire's account. He begins gently with a very simple series, based on a deck of cards. Suppose the top card is moved over the edge of the pack to its maximum overhang without overbalancing, obviously half its length, what about the next one down? How far can it be moved without toppling both cards? The series expressing this stable progressive shift of the top 51 of the 52 cards is:

1/2 + 1/4 + 1/6 + 1/8 + 1/10 + 1/12 + 1/14 + 1/16 + … + 1/102

Now, in J notation, for the first eight terms this is:

```
   NB. Reciprocals of twice the first  8 integers, excluding zero:
   % 2 * 1 + i. 8
0.5 0.25 0.166667 0.125 0.1 0.0833333 0.0714286 0.0625
```

and for a pack of 52 cards:

```
   NB. Sum of reciprocals of numbers 2 through 102:
+/ % 2 * 1 + i. 51
2.25941
   NB. (or 2.2594065907333398 to the maximum precision available)
```

The original expression can be rewritten as:

$1/2 \times (1/2 + 1/3 + 1/4 + 1/5 + 1/6 + 1/7 + 1/8 + \ldots + 1/51)$

Or, as J has it, the sum:

```
   +/ 0.5 * % 1 + i. 51
2.25941
```

This, in other contexts is called the *harmonic series* and it grows without limit: it is divergent. It becomes apparent that this series was carefully chosen, as later analyses of the zeta function centre around infinite series similar to the harmonic series but convergent. Riemann's Hypothesis reads All non-trivial zeros of the zeta function have real part one-half, so means had to be found to extend the domain of the zeta function over the whole complex plane and in the course of indicating how this is possible, while avoiding heavy calculus, Derbyshire derives series which do yield values less than 1. He shows that:

$\log(1 - x) = -x - (x^2/2) - (x^3/3) - {}^4/4) - (x^5/5) - (x^6/6) - (x^7/7) - \ldots$

is true when $x = -1$. In fact, it is equivalent to:

$\log 2 = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 - \ldots$

which resembles the harmonic series but is convergent.

```
   NB. if we give it enough terms it is log 2.
   +/_1* (_1^1+i.1000000)%1+i.1000000
0.693147
```

Now the log 2 expression can be given in J by

```
    +/1,_1r2,1r3,_1r4,1r5,_1r6,1r7
 319r420
   +/_1*(_1^1+i.7)%1+i.7 NB. not that 7 terms gets us very far!
 0.759524
```

But Derbyshire, having introduced the log 2 series, makes an apparently casual observation:

> It is, in fact, a textbook example of the trickiness of infinite series. It converges to log 2, which is 0.693147180559453 …
>
> *but only if you add up the terms in this order.* If you add them up in a different order, the series might converge to something different; or it might not converge at all!
>
> … Convergent series fall into two categories: those that have this property and those that don't. Series like this one whose limit depends on the order in which they are summed, are called 'conditionally convergent'.

Unfortunately at this stage in his narrative he does not offer any clues as to why and how the order of a series is liable to change. It seems that I must first come to terms with the second part of the book where a real 'for instance' is promised. However, he does supply an example of what is meant, by first amending the order of the terms of the log 2 expression, then enclosing some terms in parentheses and resolving the parentheses thus:

> 1 − 1/2 − 1/4 +1/3 − 1/6 − 1/8 + 1/5 − 1/10 − …
>
> Just putting in some parentheses, it is equal to
>
> (1 − 1/2) − 1/4 +(1/3 − 1/6) − 1/8 + (1/5 − 1/10) − …
>
> If you now resolve the parentheses, this is
>
> 1/2 − 1/4 + 1/6 − 1/8 + 1/10 − … ,
>
> which is to say
>
> 1/2 (1 − 1/2 +1/3 − 1/4 + 1/5 − …).
>
> The series thus rearranged adds up to one-half of the un-rearranged series!
>
> This, as the saying goes, "gives one furiously to think"!

These expressions can be put into J and will be easier to follow if we use J notation for negative numbers, which touches on a topic about which Cornelius

Lanczos had quite a lot to say and will be sympathetically received by *Vector* readers, as this quotation from his *Numbers Without End* will show: (p88)

> If we think of the picture in which we count steps, 12 − 15 means that we should go forward 12 steps and backward 15 steps. This will bring us three steps *to the left* from zero, thus generating new points which did not exist before… Our usual emotional associations with the words *positive* and *negative* are here positively out of place. The complete symmetry of the two halves of a straight line demonstrates that negative numbers are in no way inferior to positive numbers and can be employed with the same justification.

Lanczos suggests that this parity could be emphasised by indicating a positive number with a superscribed → and a negative number with superscribed ← . He goes on:

> Our usual notation is less fortunate. We write + 3 and − 3 which gives the impression that the same number 3 is once added, once subtracted. But *minus 3* does not mean that we should subtract 3. The minus sign belongs to the digit 3 and designates a new number, created by the operation of subtracting 3 from 0.

He notes that the Hindus were aware of this distinction and marked negative numbers with a superscript dot. He has proposed instead to use an overline thus:

$0 - 3 = \overline{3}$ and generally: $0 - a = \overline{a}$

Thus to add a positive number is the same as to subtract the corresponding negative number and to subtract a positive number is the same as to add the corresponding negative number. J emphasises this by a distinction between the primitive *verb* `-` (subtract) and the underbar `_`which is an intrinsic part of any negative number: so `5 - _3 = 5 + 3` and `5 - 3 = 5 + _3`. This allows parentheses and sequence changes to be resolved without the ambiguity of 'the rule of signs'. This clarifies Derbyshire's example of series re-arrangement and led me to appeal to Graham in these terms:

> Dear Graham, (26 March)
>
> Please can you help me clear my head.
>
> Derbyshire quotes the series: log 2 = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 - … of which he says:

> It is, in fact, a textbook example of the trickiness of infinite series. It converges to log 2, which is 0.693147180559453 … but only if you add up the terms in this order. If you add them up in a different order, the series might converge to something different; or it might not converge at all !

He demonstrates using this extract from the series, saying:

> 1 - 1/2 - 1/4 + 1/3 - 1/6 - 1/8 + 1/5 - 1/10 - … Just putting in some parentheses, it is equal to (1 - 1/2) - 1/4 +(1/3 - 1/6) - 1/8 + (1/5 - 1/10) - … If you now resolve the parentheses, this is 1/2 - 1/4 + 1/6 - 1/8 + 1/10 - … , which is to say 1/2 (1 - 1/2 + 1/3 - 1/4 + 1/5 - …). The series thus rearranged adds up to one-half of the un-rearranged series !

Now Cornelius Lanczos points out that the Hindus distinguished a negative from a positive number by putting a dot over it, …

and after quoting Lanczos I continue …

So Derbyshire's example above can be written in J

```
   NB. with parentheses as above:
   (1+_1r2)+_1r4+(1r3+_1r6)+_1r8+(1r5+_1r10)
47r120
   NB. without parentheses:
   1+_1r2+_1r4+1r3+_1r6+_1r8+1r5+_1r10
47r120
   NB. as sum of a list of numbers:
   +/1,_1r2,_1r4,1r3,_1r6,_1r8,1r5,_1r10
47r120
   NB. same, in descending order:
   +/1,_1r2,1r3,_1r4,1r5,_1r6,_1r8,_1r10
47r120
   NB. D. says this is half the amount!
   1r2*(1+_1r2+1r3+_1r4+1r5) 47r120
```

The `+/` expressions emphasise that we are talking throughout about summation; it just happens that some of the values are negative. With respect to convergence Anthony suggested that I consult his old 1944 paperback by Eugene Northrop, *Riddles in Mathematics* and there I found an almost identical account of the effect of inserting parentheses into the log 2 series in his chapter called *Paradoxes of the Infinite*. This leaves me even more confused, because his explanation reads:

> The difficulty arises from our attempt to apply to infinite series the processes of finite arithmetic. In finite arithmetic we go on the assumption that we can insert and remove brackets at will, grouping terms in any way we please. In other words, we assume that
>
> A + B + C = (A + B) + C = A + (B + C).

But surely the point that Lanczos made so cogently is that the associative and commutative laws do not apply to subtraction … What am I misunderstanding?

To which Graham responded: (27 March)

I think you're misunderstanding the infinity bit a little! There is no way, of course, that Derbyshire is saying "this is half the amount" at the following stage:

```
   NB. D. says this is half the amount!
   1r2*(1+_1r2+1r3+_1r4+1r5)
47r120
```

To illustrate in J what Derbyshire is saying we need to make two calculations: take n terms of the original series, once in the order they originally come in (the first n terms) and once in Derbyshire's order.

I love the series and its zany characteristic! I don't remember seeing it before.

Let's look at `2*n` terms of the original series

```
   series =: 3 :',1 _1*&.:|:%(y,2)$>:i.+:y'
   [s =: series 8r1
 1 _1r2 1r3 _1r4 1r5 _1r6 1r7 _1r8 1r9 _1r10 1r11 _1r12 1r13
 _1r14 1r15 _1r16
```

Indices of the first n terms of this series are `i.n`

Indices of n terms of `Derbyshire's` series are `ind n` where

```
   ind =: 3 :'/:~(+:i.>.y%3),>:+:i.0>.<.2r3*y'
   ind 20
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 15 17 19 21 23 25
```

For n = 13, the indices for s in each case are:

```
      (i.,:ind)13
 0 1 2 3 4 5 6 7 8 9 10 11 12
 0 1 2 3 4 5 6 7 8 9 11 13 15
```

giving these two series:

```
      s{~(i.,:ind)13
 1 _1r2 1r3 _1r4 1r5 _1r6 1r7 _1r8 1r9 _1r10  1r11 _1r12  1r13
 1 _1r2 1r3 _1r4 1r5 _1r6 1r7 _1r8 1r9 _1r10 _1r12 _1r14 _1r16
```

Summing them we get:

```
      8j4":+/"1 s{~(i.,:ind)13
 0.7301  0.4284
```

We can observe the factor of 2 beginning to appear for n = 13. As n increases so the two answers would be expected to converge to `^.2` and `-:^.2`.

To which I responded: (28 March 15:14)

Of course I agree with everything you say, but that is not my problem. Derbyshire is not saying that the value of an accumulation of an extract from an infinite series will most probably not match the accumulation of a different extract, that is so obvious as not to need saying, I hope. The example he quotes does not use two different sets taken from the series. What he is talking about is the sequence in which they are accumulated and he effects this change by enclosing some pairs of terms in parentheses, evaluating the parentheses and then accumulating his partial result. I think his 1/2 (1 - 1/2 + 1/3 - 1/4 + 1/5 - ...) suggestion is just an abuse of algebra. Let us use the first 10 terms of the original alternating series without confusing the issue with parentheses and then merely change the sequence. This is what he is actually suggesting is critical after all. Then we have:

```
    NB. first 10 terms but remember this is J:
    1-1r2+1r3-1r4+1r5-1r6+1r7-1r8+1r9-1r10
 1117r2520
    NB. rearranged & sure enough a different result:
    1+1r7-1r8+1r9-1r10-1r2+1r3-1r4+1r5-1r6
 1151r2520
    NB. now taking Lanczos to heart:
    1+_1r2+1r3+_1r4+1r5+_1r6+1r7+_1r8+1r9+_1r10
 1627r2520
```

```
   NB. re-arrangement has no effect:
   1+1r7+_1r8+1r9+_1r10+_1r2+1r3+_1r4+1r5+_1r6
1627r2520
```

What we must remember is that J does not evaluate the same way as conventional maths, which does it this way:

```
   (((((((((1-1r2)+1r3)-1r4)+1r5)-1r6)+1r7)-1r8)+1r9)-1r10)
1627r2520
```

and this way there is no problem with the *sequence*

```
   (((((((((1+1r7)-1r8)+1r9)-1r10)-1r2)+1r3)-1r4)+1r5)-1r6)
1627r2520
```

but, of course, using different sets is unlikely to give the same answer. It is not the sequence of terms that is the problem, it is the sequence of evaluation if `-` is confused with `_` … I think

To which Graham replied: (28 March 20:28)

Aren't there two quite separate issues here? One is notational and the other is the infinite series paradox. The notational one will get anyone into trouble who doesn't follow the rules on whatever issue they tackle, but I see no reason to believe Derbyshire hasn't followed the rules. But, from your last sentence, it seems you think he has.

Where does he make a *notational* error when deriving his 1/2(1 - 1/2 + 1/3 - 1/4 + ...)?

The reason for the paradox is exactly that the value of an accumulation of an extract from an infinite series will most probably not match the accumulation of a different extract. We are interested in the series

1 + -1/2 + 1/3 + -1/4 + 1/5 + -1/6 + -1/7 + -1/8 + ...

If you begin taking these terms in the order Derbyshire does, before bracketing them up, 2/3 of the terms he extracts are negative numbers and only 1/3 are positive. No wonder it converges to a different sum!

Which provoked me to: (30 March 14.59)

Thank you for keeping me thinking – what I want to say to your last is … yes, but … what are these mathematicians agonising over? What is it that they find so remarkable? If we are allowed to pick and choose among the terms of an infinite series to be summed, we have an infinity of results to choose from. If we use only the positive terms our sum diverges:

```
   +/\1r3,1r5,1r7,1r9,1r11
0j6":1r3 8r15 71r105 248r315 3043r3465
0.333333 0.533333 0.676190 0.787302 0.878211
```

If we choose the negative terms, our sum diverges negatively:

```
   0j6":+/\_1r2,_1r4,_1r6,_1r8,_1r10
_0.500000 _0.750000 _0.916667 _1.041667 _1.141667
```

If we modify an initial positive, we go from positive to negative:

```
   0j6":+/\1,_1r2,_1r4,_1r6,_1r8,_1r10
1.000000 0.500000 0.250000 0.083333 _0.041667 _0.141667
```

A judicial mixture of positive and negative will get us somewhere near the number we first thought of – but what is the point? What conclusion are we being asked to draw? I thought that there must be some argument to suggest that the log 2 irrational is efficiently approximated by the summation of the alternative. There may be other values as good or better to be obtained by permutations of the terms, but these 'Gee-Whiz' rearrangements seem frivolous.

Here it seems justifiable to select only the first 60 terms:

```
   NB. alternating sum of reciprocal of 2 to powers 0 to 6:
   - /% 2 ^ i.7
0.671875
   NB. and here it is for 53 terms at maximum precision:
   - /% 2 ^ i.53
0.66666666666666674
   NB. while 60 terms reduces it a bit …:
   - /% 2 ^ i.60
0.66666666666666663
   NB. 90 shows no change, so two-thirds looks like the limit:
   - /% 2 ^ i.90
0.66666666666666663
```

Is there something here other than a statement of the blindingly obvious?

To which Graham came back: (30 March 18:03)

OK! So what is all the fuss about?

We love infinite series, especially ones that converge, because they sum to a number that is unique and a number that cannot always be found exactly. For example I don't remember (if I ever did know) how to demonstrate our series converges to log 2. So there is mathematical excitement associated with infinite series. An infinite series is defined by its first few terms, assuming the continuing sequence is unambiguous, so the shocking thing about our sequence is that you can clearly write it down two different ways and so obviously get two very different answers! The fact that you are reordering the series is not immediately apparent.

Another point that makes this a special series is that it converges very very slowly. I think I am right in saying that there are very many infinite series whose sum would not be affected by the alteration in order that was made to our series. Obviously in the extreme case when only positive terms are taken from an alternating series you will get a different answer, but there are many series that converge to the same sum irrespective of how you order its tail end.

Sylvia to Graham (1 April 16:41) — now we both have our teeth in it!

The log 2 series is a method of approximating the irrational by accumulating values which decrease according to a clear pattern, either positively or negatively — a sort of Lambeth Walk. Thus the adjustments continually decrease in significance until we call a halt, for the very practical reason that we cannot work with infinite expansions. My version of J calls log2, 0.693147 and half log2, 0.346574, when in standard display mode.

You chose two sets of terms from the log 2 series. They have the first ten in common so we can keep the display small by displaying the sum of those as the first cumulative value:

```
   8j4":+/1 _1r2 1r3 _1r4 1r5 _1r6 1r7 _1r8 1r9 _1r10
 0.6456
```

You then arrive at two different cumulative results from your two chosen sets of 13 terms:

```
   8j4":+/\0.6456 1r11 _1r12 1r13
 0.6456 0.7365 0.6532 0.7301
   8j4":+/\0.6456 _1r12 _1r14 _1r16
 0.6456 0.5623 0.4908 0.4283
```

You then posit the case that as the number of terms increases 0.7301 will tend to 0.693147 while 0.4283 will tend to 0.346574. What is not clear is how the additional terms are to be chosen. You have used two selections from the first 16 so we could decide to ignore those and take say a further 8 from 16 onwards:

```
   8j4":+/\0.7301 1r17 _1r18 1r19 _1r20 1r21 _1r22 1r23 _1r24
0.7301 0.7889 0.7334 0.7860 0.7360 0.7836 0.7382 0.7816 0.7400
   8j4":+/\0.4284 1r17 _1r18 1r19 _1r20 1r21 _1r22 1r23 _1r24
0.4284 0.4872 0.4317 0.4843 0.4343 0.4819 0.4365 0.4799 0.4383
```

So far both values have increased and this is getting tedious, given that we are left with infinity minus 24 to add.

So what should we do about the terms we have so far left out of each set. What happens if we put them back in?

```
   8j4":+/\0.7400 _1r14 1r15 _1r16
0.7400 0.6686 0.7352 0.6727
   8j4":+/\0.4383 1r11 1r13 1r15
0.4383 0.5292 0.6061 0.6728
```

The difference, unsurprisingly, is accounted for by the terms we left out. We have thus been justified in our belief that A+B+C = (A+B)+C = A+(B+C) or even C+B+A and all other permutations, but A+B does not equal A+C unless A=B=C or A=0. What we have illustrated is that the sequence of the series is immaterial providing that all terms up to a selected one are represented. I think you are right to say of this so-called “paradox of the infinite” that “this only has the appearance of significance”.

Moreover, I think Derbyshire goes a step too far when he says:

> 1 - 1/2 - 1/4 +1/3 - 1/6 - 1/8 + 1/5 - 1/10 - …
>
> Just putting in some parentheses, it is equal to
>
> (1 - 1/2) - 1/4 +(1/3 - 1/6) - 1/8 + (1/5 - 1/10) - …
>
> If you now resolve the parentheses, this is
>
> 1/2 - 1/4 + 1/6 - 1/8 + 1/10 - … ,
>
> which is to say
>
> 1/2 (1 - 1/2 + 1/3 - 1/4 + 1/5 - …).

> The series thus rearranged adds up to one-half of the un-rearranged series!

Oh yeah!

```
   NB. rearranged and two terms missing, ... but so what.
   +/1 _1r2 _1r4 1r3 _1r6 _1r8 1r5 _1r10
47r120
   NB. resolve some parentheses
   (1+ _1r2), _1r4, (1r3+ _1r6), _1r8, (1r5+ _1r10)
1r2 _1r4 1r6 _1r8 1r10
   NB. add up the result. No change so far.
   +/1r2 _1r4 1r6 _1r8 1r10
47r120
   NB. extracting one-half ...
   (1r2 _1r4 1r6 _1r8 1r10)%1r2
1 _1r2 1r3 _1r4 1r5
   NB. doubles the result …
   +/1 _1r2 1r3 _1r4 1r5
47r60
   NB. so we must times half to get back where we started
   1r2*1 _1r2 1r3 _1r4 1r5
1r2 _1r4 1r6 _1r8 1r10
   +/1r2 _1r4 1r6 _1r8 1r10
47r120
   NB. sure is equal!
   (+/1 _1r2 _1r4 1r3 _1r6 _1r8 1r5 _1r10) = (+/1r2*1 _1r2 1r3
_1r4 1r5)
1
```

But what could possibly justify us writing it this way?

```
(+/1 _1r2 _1r4 1r3 _1r6 _1r8 1r5 _1r10 … ) = (+/1r2*1 _1r2 1r3
_1r4 1r5 …)
```

What could this mean? How are these apparent series to be continued? Is the next term positive or negative? I still think it’s an abuse of algebra and uninformative into the bargain.

Stroppy Sylv

Graham's response: (1 April 21:35)

You: The log 2 series is a method of approximating the irrational by accumulating values which decrease according to a clear pattern, either positively or negatively – a sort of Lambeth Walk. Thus the adjustments continually decrease in significance until we call a halt, for the very practical reason that we cannot work with infinite expansions.

Why call a halt? Only if we are computing a result term by term. But the equality

Log 2 = 1 + -1/2 + 1/3 + -1/4 + …

must not be halted. Halting it makes it no longer correct.

You: What is not clear is how the additional terms are to be chosen.

I think you may have missed a key point here. There is a simple rule for determining additional terms, which I gave in my first reply, i.e.

```
   ind =: 3 :'/:~(+:i.>.y%3),>:+:i.0>.<.2r3*y'
   ind 20
0 1 2 3 4 5 6 7 8 9 10 11 12 13 15 17 19 21 23 25
```

Using the verb `ind` you can find the additional terms. From the pattern above it is clear that the 21st term is either going to be 14 or 27, and it turns out to be 27. So the “rearranged” series is precisely defined for any number of terms. But note that the trailing terms are all odd and these are all negative ones, demonstrating what I said in my last email that there are twice as many negative terms as positive ones. The rearrangement is a completely different infinite series from the original. The sleight of hand is in the bracketing process which initially fooled me into thinking the original and the rearrangement were one and the same.

Are you laying your stroppiness at Derbyshire’s door? I cannot spot anything you have told me about his argument that I can fault. I think you're having a genuine struggle to see the wood for the trees here, and worthwhile struggles can be painful.

I hope this helps. Taking an equal number of terms of each is necessary to keep up the paradox because if you took an arbitrary number of each then comparing the two sums would have no significance! Keeping them equal as you approach infinity is intriguing, but actually it only has the appearance of significance. I think the paradox simply demonstrates that the order in which you choose to sum an infinite series can sometimes affect the answer.

Sylvia to Graham (8 April 18:12) — so I turned my attention to the example in Northrop’s book.

Oooh! I am having fun. This is how Northrop illustrates his 'paradox of the infinite'.

L=1-1/2+1/3-1/4+1/5-1/6+1/7-1/8+… +1/13-1/14+1/15-1/16+…

Grouping terms first by twos and then by fours.

[1] L=(1-1/2)+(1/3-1/4)+(1/5-1/6)+(1/7-1/8)+… +(1/13-1/14)+(1/15-1/16)+…

[2] L=(1-1/2+1/3-1/4)+(1/5-1/6+1/7-1/8)+… +(1/13-1/14+1/15-1/16)+…

Dividing both sides of equation [1] by 2 we get

[3] 1/2L=(1/2-1/4)+(1/6-1/8)+(1/10-1/12)+(1/14-1/16)+…

Adding, bracket by bracket, equations [2] and [3],

[4] 3/2L=(1+1/3-1/2)+(1/5+1/7-1/4)+(1/9+1/11-1/6)+(1/13+1/15-1/8)+ …

which is obviously, he says, equal to

1-1/2+1/3-1/4+1/5-1/6+1/7-1/8+1/9-1/10+1/11 …

The trick is not at all obvious when step [4] is glossed over so confidently: The expression [4] was arrived at by adding the two-term brackets from [3] to the four-term brackets from [2] and then simplifying; e.g. the first bracket is the result of working out 1-1/2+1/3-1/4+1/2-1/4. You see how the -1/2 and +1/2 terms cancel out and -1/4+-1/4 becomes -1/2 to take the place of the lost term. So, in the same way, the terms metamorphose into each other until all that we are left with is the original alternative L which has miraculously been proved by simple arithmetic to be equal to 3/2L!

But notice that when we added the first set of brackets we used a positive term +1/2 to get rid of the -1/2 and, very conveniently, the duplicated -1/4 terms converted themselves into the, now cancelled out, -1/2 term. Talk about smoke and mirrors! This is why I went to my mathematical dictionary to see just what is meant by *monotonically decreasing*, which expression [4] is manifestly not doing, and then to check on the meaning of *alternative* and *conditionally convergent*. It has been a fascinating journey and without you to goad me I might never have taken it. Thank you.

Monotone Monotonic adj… .

A monotonic decreasing quantity is a quantity which never increases.

The use of “never” here suggests that the order of the absolute value of the terms is significant, but for summations I think it is always irrelevant except perhaps for practical questions about the inevitable limitation of calculation to a finite number of terms, which might make it invalid to use terms taken almost exclusively from way out along an infinite series.

Alternating adj.

… alternating series. A series whose terms are alternately positive and negative, as

1 - 1/2 + 1/3 - 1/4 + (-1)n-1/n + … .

An alternating series converges if the absolute values of its terms decreases monotonically with limit zero (the Leibnitz test for convergence). This is a sufficient but not a necessary condition for convergence of an alternating series. If one convergent series has only positive terms and another only negative terms, then the series obtained by alternating terms from these series is convergent, but the absolute values of its terms may not be monotonically decreasing. The series

1 - 1/2 + 1/3 - 1/4 + **1/9 - 1/8**+ 1/27 - 1/36 + … is such a series.

This it seems would guarantee log2 convergence, but see below:

Convergence n… .

Conditional convergence An infinite series is conditionally convergent if it is convergent and there is another series which is divergent and which is such that each term of each series is also a term of the other series (the second series is said to be derived from the first by a rearrangement of terms); i.e.; an infinite series is conditionally convergent if its convergence depends on the order in which the terms are written. A convergent series is conditionally convergent if and only if it is not absolutely convergent. E.g.; the series 1 - 1/2 + 1/3 - 1/4 + … is conditionally convergent because it converges and the series 1 + 1/2 + 1/3 +… diverges.

For a summation to depend upon the order in which the terms are written, when combining it with terms from another series having just the same set of terms, can only be by changing some + and − signs; but this would mean that the series is not a summation but a series of arguments to addition and subtraction functions. If, taking Lanczos to heart, we use a notation which distinguishes a negative number from the positive argument to a subtraction function, the alternating series for log2, for instance, contains only negative 1/2 and positive 1/3. But negative 1/2 is not a member of the divergent series 1+1/2+1/3 … referred to above as the "second series said to be derived from the first by a rearrangement of terms".

Now, by the definition above, the *alternating series* for log2, contains an infinity of reciprocals of positive odd integers and an infinity of reciprocals of negative even integers. It does not contain positive even or negative odd denominators although there is, of course, an infinite series having wholly negative terms which is monotonically divergent and complements the other infinite divergent series 1+1/2+1/3+ … as mentioned in definition 3 above. Now, which one of these 2 divergent series bears the relationship to the log2 *alternating series*defined above, "such that each term of each series is also a term of the other series (the second series is said to be derived from the first by a rearrangement of terms"?

It is easy to prove that the log2 alternate is insensitive to parentheses if it is written using J notation:

```
   +/1 _1r2 1r3 _1r4 1r5 _1r6 1r7 _1r8 1r9 _1r10 1r11 _1r12
18107r27720
   +/(1 _1r2),(1r3 _1r4),(1r5 _1r6),(1r7 _1r8),(1r9 _1r10),(1r11
_1r12)
18107r27720
   +/(+/1 _1r2 1r3 _1r4),(+/1r5 _1r6 1r7 _1r8),(+/1r9 _1r10 1r11
_1r12)
18107r27720
```

Order of summation changes nothing, the total depends on the number of monotonically decreasing terms taken. Division, however, creates new terms of which some are in the log2 series and some from a different and divergent series and this applies irrespective of parentheses:

```
   1r2*1 _1r2 1r3 _1r4 1r5 _1r6 1r7 _1r8 1r9 _1r10 1r11 _1r12
1r2 _1r4 1r6 _1r8 1r10 _1r12 1r14 _1r16 1r18 _1r20 1r22 _1r24
```

The following terms are in the positive divergent series but excluded from the log2 series:

```
   NB. are not in log2 alternating series:
   8j4":+/1r2 1r6 1r10 1r14 1r18 1r22
   0.9391
```

These terms are already in the log2 alternating series and do not occur twice:

```
   NB. are in the log2 alternating series
   8j4":+/_1r4 _1r8 _1r12 _1r16 _1r20 _1r24 _0.6125
   NB. which adds to half log2
   NB. if we add the terms found only in the divergent series
   _0.6125+0.9391
0.3266
   NB. so their combined sum is indeed
   NB. half the total of the original alternating series:
   0.3266*2
0.6532
```

This is the basis on which Northrop claims that *rearrangement* of its infinite series can demonstrate that the log2 series is equal to 1.5 times the log2 series, which should sum to:

```
   8j4":18107r27720 NB. total of first 12 terms
0.6532
```

So by doing some simple arithmetic, terms from two series (one alternating and another divergent) have been intercalated with the apparent effect of adding half as much again to the total. This is described as a mere *rearrangement* of the alternating series so, as usual with paradoxes, it all boils down to a question of terminology: "when is a member of a series not a member of a series?" Answer, "only when you can't think of an algorithm which would convert a non-member to a *bona fide* member". In other words, the term *rearrangement* covers the whole panoply of algebraic manipulation. It is not merely a question of the*order* in which the terms are taken — in fact the order is immaterial providing sequence A contains the same selection of *either* positive or negative terms as sequence B… .

Only a bit breathless,

Sylv

In response to which Graham sent: (14 April 21:01)

> Dear Sylv,
>
> You’re still not happy with it, are you? The bracketing is fine, and I like your `alternative` method. But I prefer the original bracketing method which is nicely visualised by the table I have enclosed in the attached Word document. And I think this table helps to illustrate why the two infinite series are not equal. But see what you think!

```
      1     1r2   1r3   1r4   1r5   1r6   1r7   1r8   1r9   1r10  1r11  1r12  1r13
1r2   1     _1r2
_1r4                    _1r4
1r6               1r3               1r6
_1r8                                            _1r8
1r10                          1r5                           _1r10
_1r12                                                                   _1r12
```

> This `table` represents the top left hand corner of an *infinite* table. Along the top line we have our first infinite series ‘properly’ ordered. Down the left column we have our second infinite series ‘properly’ ordered. Next, all the terms of our first *infinite* series are distributed to the body of the table, each being dropped to a row according to the linear pattern clearly visible: the filled places form two straight sloping lines. Adding along the rows we get the result shown in the left column, i.e. the second *infinite* series.
>
> This is a pictorial equivalent of the bracketing and what I used to derive that early J expression I gave you defining the reordering of the terms. It gives the impression that the two *infinite* series are equal, but they are not: the reordering causes the negative terms to be used up twice as quickly as the positive ones, so no wonder the sum is less.
>
> Notice I have italicised *infinite*! There are issues associated with the infinite and the infinitesimal that cannot readily be replicated in finite examples. For example, take a finite table of the kind shown above and you get an incomplete first series equal to an incomplete second series – no great deal, as you have noted in earlier emails.

This is the point to which Graham and I have arrived to date. I am hoping some others of the APL/J community might join the fray. Do we have any takers?

Subscribing to Vector

Your *Vector* subscription includes membership of the British APL Association, which is open to anyone interested in APL or related languages. The membership year runs from 1 May to 30 April.

Name	________________________
Address	________________________
	________________________
Postcode/Zip and country	________________________
Telephone number	________________________
Email address	________________________

UK private membership	£20	__
Overseas private membership	£22	__
+ airmail supplement outside Europe	£4	__
UK corporate membership	£100	__
Overseas corporate membership	£110	__
Non-voting UK member (student/OAP/unemployed)	£10	__

Payment methods (*Sterling only)*

1. A Sterling cheque, payable to *British APL Association*, drawn on a UK bank.
2. By American Express, MasterCard or Visa:

I authorize you to debit my American Express/MasterCard/Visa account

Number: ____________________ Expires: ___/___

for the membership category indicated above.

Signature: ____________________ Date: ________

Privacy Policy

Your personal information will be stored on computer but not disclosed to third parties. Card data will not be stored on computer.

3. By electronic transfer.

Our account details are: Barclay's Bank; Cambridge, Chesterton Branch; Sort code: 20-17-35; Account number: 63955591; Account name: British APL Association; SWIFTBIC: BARCGB22; IBAN: GB86 BARC 2017 3563 9555 91.

4. Use PayPal to credit account treasurer@vector.org.uk (no account needed – ask for details).

If you pay by cheque or credit card, please send the completed form to:
BAA, c/o Nicholas Small, 12 Cambridge Road, Waterbeach, Cambridge CB25 9NJ