

THE STEAM-HAMMER AND THE FLY

From: Gérard A. Langlet

A French proverb says: "Never use a steam-hammer to smash a fly". It (or its English equivalent counterpart) should be repeated on every page of any book or tutorial devoted to APL.

Vigorously attacked in Vector 7.1 [1], I shall answer in a way that may not fulfill Bob Bykerk's awaitings completely, but as much as I can in a few pages; indeed a whole book would be better - that will come in the future, with a general survey of what APL led me to as a *tool of thought*.

First, I have been programming in APL for about 20 years, writing sophisticated applications for research and industry, i.e. some important subset of the "real-world". Quite often, I also have to translate my algorithms into other languages (C, Fortran, Pascal inter alia) for many various reasons: execution speed, compatibility with other existing programs or systems, absence of APL implementations (Cray II, parallel machines), hatred or incompetence of colleagues about APL, or simply the *still-high cost* of good APL interpreters or some hardware. The real world has many many constraints of all kinds..., so, when programming in APL, which still is the best language ever designed to test a new idea and build a prototype, it does not seem to me superfluous to think of what will come next. The "RISC" programming style is the fruit of hard experiments. It may not correspond to what is taught in manuals about data structures and/or the usage of APL notation. But I am nothing of a masochist, and I would gently conform to any other so-called "healthy coding style" if I were convinced of its real superiority. APL should not remain like an ivory tower, the real world is indeed a multi-language environment, APL being however the best tool for *thinking & simplifying*.

Many people often believe that the beauty and the expressive power of the APL notation on the paper, especially in extended implementations, will lead them to write easily nice and efficient code. Just try $R \leftarrow V$ with e.g. $V \leftarrow 10000 \rho 3 4 5 0 2 3$ on some APL2-compatible interpreters (or: ρ, V if monadic ρ "ENLIST" is not available), and compare with:

$R \leftarrow V \rightarrow + \setminus V \diamond R \leftarrow R + \uparrow \rho R$ or - if you just hate the diamond - with:
 $R \leftarrow R + \uparrow \rho R \leftarrow V \rightarrow + \setminus V$ (but beware of WS FULL).

You may read [2] if you want to know more of what happens and why.

Let us talk now about data structures; one is taught to represent sales data in 3 dimensions: salesman \times month \times year. Why not, if you just want to use the power of $+ /$ on any of the major axes? But this is in fact a simplistic case; one might want to extract much more sophisticated information from this array - see below. And what happens if the data contain incomplete information, (jokers, see [3]), if it is not a parrallelepipedic array (there is an important conceptual difference between an appointed salesman who has sold nothing because he went fishing, and another one who could not sell anything for he had not joined the company yet or for he retired or went sick a few months ago!)?

There are many more cases for which the data structures provided by APL, APL2 or other languages such as LISP and Prolog, do not suffice to cover the real needs of the mathematical if not of the real world. LISP offers circular lists, Prolog3 offers infinite recursive trees, but we need graphs... How can you represent the topology of a simple cube? How many faces has a cube? 6 (after what you were taught at school). But just think a bit: if you describe a polyhedron with only 5 faces, isn't the sixth one implicit? then, if so, which face is implicit among the six ones? Then, you must realize that the description of the cube is a simple trivial problem if you compare it with the extended 3-D chemical formula of morphine for which nobody is able to tell how many cycles it contains! Unfortunately, morphine belongs to the real world; and you have to think of it when you try to build in APL a structure-retrieval data base with thousands of compounds.

Another example will be given by natural language processing. What is the ideal structure for a page of text? Is it an array with rows and columns, a vector of characters with embedded carriage-returns, a nested vector of words some of which may be cut by an optional "caesura"?

In fact, it depends on what you want to do with the text. Conceptually, the text is all that AT THE SAME TIME, and many more potential structurations are possible (e.g. sentences, groups of words). Also remain conscious that the text may be imperfect, containing typos, spelling or construction errors, and may refer to elliptic contextual ideas or require previous knowledge of the reader about the subject.

In most cases, the *ideal* data structure has not been discovered yet. It is certainly not an array. It may be fractal, cf [9,10]. But let us come back to APL which, in some implementations (e.g. APL68000, APL90, and, to a certain extent, Dyalog-APL and APL*PLUS), offers fast reversible MATRIX \leftrightarrow VECTOR conversion tools such as DBOX. In general, a text page is shorter when kept as a vector. This is also true for numeric data with heterogeneous length, but DBOX accepts them only in the first 2 implementations. Most of my data are kept in vectors, and if and only if it is necessary to handle them in matrices or in vectors of vectors or in vector of matrices, they are TEMPORARILY, rapidly and internally reshaped with automatic mechanisms which accept ANY of these possible data structures as input. Another "philosophical" example of this is the generalised "EXECUTE" function; it is rather object-oriented: it will try to execute anything as its argument: an APL expression or function as well as a BASIC-, C-, Pascal-, FORTRAN- or machine-code. It is also able to recognise expert-system rules, some commands in natural languages or in DOS or UNIX as well as Debussy's or Mozart's music, and graphics. It does not yet play chess or accept medical prescriptions, but why not, in the future real world of year 2001 (Guess who invented "Ho!")?

Another item of my short letter to Vector [4] has been misunderstood. I never wrote not to use locals; I think that locals should be avoided when they are unnecessary, except e.g. for voluntary didactic purposes. I would appreciate an extended APL-implementation in which all one-character names would be automatically localised; this would save typing time and space. Since I deprecate the use of parentheses, APL expressions are shorter and do not lead so frequently to the WS FULL message; then, I do collect the intermediate results within the minimum of locals. With "RISC" programming style, I don't burn my bridges. My programs are easier to debug than before. Nobody is obliged to believe me; just try, if you wish, and criticise only in 6 months from now. The first time you drank whisky, did you really appreciate? So many people, inter alia E. Dijkstra, have already warned people about the dangers of APL programming..., but I now admit he was right when he promoted structured programming, wanting to kill the *dinausuresque* "goto"; see also [11].

Try to rewrite ex nihilo with branches, the PERM function listed in [5]. It is not an easy way of doing it. Then, try to write a screen manager using some combinations of the 3 buttons of a mouse as well as the keypad, in order to handle several recursive pop-up menu-windows with lifts, travelators, automatic clipping, shadows and help files. I did that once, in APL*PLUS PC, and I doubt that I could have succeeded with branches and labels [6]. The result is a short general-purpose RISC-APL function which respects several other rules of my anathematised programming habits: the display of a function should not exceed the screen size, so that you never have to scroll up or down to see what it does; every procedure (simply a character string or vector) is defined before the one which uses it; the resulting code executes quickly in APL and can be translated easily i.e. by hand or with another small program into any of the important programming languages of the real world. One may burn me, Joan of Arch and Giordano Bruno were reduced to ashes, Galileo Galilei escaped retracting himself:

"Eppur, si muove". Dear Bob, I persist and sign.