Array-Oriented Programming in Education

Richard D. Neidinger
Davidson College
P. O. Box 1719
Davidson, NC 28036
E-mail: `rineidinger@apollo.davidson.edu`

Today's campuses (college, university, and secondary school) are ready to make widespread use of array-oriented programming, in a way that a decade ago was known only to APL programmers. Array-oriented programming uses powerful built-in functions for array and matrix manipulation in an interactive workspace of user-defined functions and variables. While the tools for such programming are now widely available, the style is still not well known. This is where APL programmers can make a valuable contribution. Schools are clamoring for ways to use the new powerful computing tools, as evidenced by the massive amount of work in calculus reform in the last five years. In the future, array-oriented programming may be especially important as demand grows for (portable) algorithms that can take advantage of parallel computing. Since the potential for this tool of thought is so great, this article is the first of a series discussing how academia can use array-oriented programming in APL and in other (descendant) environments.

A simple example may help to distinguish the array-oriented style. Consider the problem of calculating:

$$h\sum_{i=1}^{n} f(a+ih)$$

where f is a real-valued function of one real variable, a and h are real numbers, and n is a positive integer. (This is a Riemann-Sum estimate of the definite integral from a to b of $f(x)$, where $h=(b-a)\div n$.) In the most common *control structure style*, this could be written:

```
sum ← 0
for i ← 1 to n
      sum ← sum + f(a+i×h)
next i
sum ← h × sum
```

In a pure *functional style using recursion*, one could implement the following:
Define the function $\sum(f,a,h,n)$ to return

```
zero if n=0 or
h × f(a) + sum(f,a+h,h,n-1) otherwise.
```

In an *array-oriented style* using APL notation, it could be:

```
sum ← h × +/f a+h × ⍳n
```

To consider the nature of array-oriented programming beyond a simple one-liner, think of a whole workspace of related array tools, such as tools for manipulating arrays of power series coefficients.

While many modern languages (including APL) can implement all three styles, the language influences which style is more natural. Still, even when the array-oriented approach is available, it may not be considered because of the programmer's unfamiliarity.

Of course, the readers of Quote Quad are very familiar with the array-oriented style and the programming environment that makes it natural. Thus, I would like to solicit three types of articles:

1. Examination of software packages, reviewing the extent to which array-oriented programming is possible and natural. Specifically, show implementations of simple array-oriented algorithms such as the above sum program. In addition, compare features of the package (or lack of) with corresponding APL features such as workspace, supplied array functions (primitives), operators that apply scalar functions across an array, and provision for multiple function arguments (dyadic?) and results. I'm planning a first such article about Mathematica. Please contact me to suggest other software and reviewers, and to suggest other features or algorithms to compare.
2. Examples of array-oriented programming being used in education. These examples will serve both to inform ourselves and to show this style to those outside the APL community. This could be an application of APL, J, or some other array-oriented language. Of course, Quote Quad has been publishing such articles for many years.
3. Discussion of general issues concerning array-oriented programming in education. Please submit an article or a letter to the editor about any related concern or reaction. To help stir things up, I'll offer a few of my thoughts in this area.

Communicating Algorithms in APL to Outsiders

An enduring concern of mine is how we can develop our communication so that those using packages such as Mathematica or MATLAB can understand our algorithms. It is important to emphasize that I'm concerned about the reader of our code or pseudo-code and am not trying to fault APL for being hard to learn or use. No matter how simple or straight-forward, every language has quirks and "features" that take practice to use properly. (Just think of the jokes about VCR programming!) Nevertheless, I can "understand" the algorithms in well-written code or pseudo-code in many diverse languages. To implement the idea, I might have to use a different language that is more familiar to me, but the writer has communicated the algorithm. Of course, some languages are going to be harder to understand than others but I suspect that the typical programmer views APL as the most obscure of all. What can we do about this?

One idea is to use names in place of symbols for primitive functions or idioms whenever the name is widely understood and the symbol is not. You can actually program in this style if you develop a workspace of standard functions. This is different from keywords or ASCII transliteration. Since a reader could understand ←, ×, ÷, and ≤, these symbols could stay. On the other hand, the specific cases of the circle function could be named (sin, cos, etc.) and dyadic and monadic cases of the same symbol could be named differently, as in max and ceiling. There is also a third category of APL symbols that enable array-oriented programming and thus do not have names recognized by those unfamiliar with the style, such as ⍳ and reduction or scan. Explaining such symbols gets right to the heart of the matter, and is much more beneficial than explaining that APL uses a different symbol for a familiar idea. Of course, there is a lot of gray area in this scheme: when is a name or symbol considered widely understood, what names should be chosen, what functionality is lost? If there is significant interest, maybe there could be a whole article outlining such a scheme.

Another idea is to use pseudo-code if you really need a control structure. Of course, first consider if an array-oriented approach could avoid the structure. Otherwise, I see no reason why an expository article, such as those solicited in 2 above, shouldn't explain an algorithm using **IF-THEN-ELSE** or a **WHILE** loop. Any APL programmer could implement it, any programmer would understand it, and the writer would be saved the embarrassment of using goto's in public.

Finally, I'd encourage authors to use good programming style as taught in most beginning programming courses. This means using decorations that are designed for human readers including comments, spacing, indentation, descriptive names, and helpful use of UPPERCASE and lowercase. These admonitions should be familiar enough to every programmer to need no explanation. Still, somehow APL programmers feel privileged (because of old software restrictions) to ignore such standard practices as indenting loops. There are several different ways to use *case* sensitivity in APL. Personally, I like to use lowercase for scalars and scalar-valued functions, UPPERCASE for vectors or vector-valued functions, and FirstLetterCap for character strings or line labels. Many people have adopted the MicroSoft standard of FirstLetterCap for functions and procedures, lowercase for variables, and UPPERCASE for constants. Peter Naeve believes in letting the case represent the tier of development, with lowercase functions feeding into the higher-level UPPERCASE functions. In any case, pay attention to your readers. ■

Grey Codes, Towers of Hanoi, Hamiltonian Path on the N-Cube, and Chinese Rings

Prof. Leroy J. Dickey
Department of Pure Mathematics
University of Waterloo
Waterloo, Canada N2L 3G1
Tel: 519-888-4567, ext 5559
Fax: 519-725-0160
E-mail: LJDickey@UWaterloo.CA

A Colorful Problem

Not long ago, on the UseNet news group of comp.lang.apl, a question was put forward by Clifford A. Reiter of Lafayette College, in Easton, Pennsylvania <reiterc@lafcol.lafayette.edu> which generated a score of responses. His question is connected with creating:

"... a function to provide pure 'hues' when working with computer images. The matrix desired consists of Blue Green Red triples running from red, to yellow, to green through the spectrum and back to red."

He asked readers to give a short APL or J expression to create the following matrix:

```
0 0 1
0 1 1
0 1 0
1 1 0
1 0 0
1 0 1
0 0 1
```

I think of this matrix as having six fundamental rows which then cycles back on itself. A number of solutions were presented; Curtis A. Jones <jonesca@sjevm5.vnet.ibm.com> noticed that the matrix obtained by joining the first two rows