AN APL IDIOM INVENTORY

Howard A. Peelle
University of Massachusetts
Cognitive Studies of Computers in Learning
Furcolo Hall #10
Amherst, MA 01003 USA
(413) 545 - 0135

Abstract

An instrument is proposed for rating APL idioms. Scales include:

Length
⎕IO Independence
Usefulness
Efficiency
Generality
Clarity
Simplicity
Memorability
Interestingness
Elegance

This APL Idiom Iventory was pilot-tested by APL programmers and APL instructors who rated a dozen selected APL idioms. The results indicate which idioms they think are "useful", "easy to learn", "hard to remember", "interesting", etc. Implications for teaching and related issues are also discussed.

Introduction

APL programmers use some APL idioms* but not others. Why? Is it because a certain idiom is short? ⎕IO-independent? efficient? clear? easy to remember? elegant? Just what are the important qualities of idioms? How can different idioms be judged, anyway?

It is understandable why the APL community has not addressed these questions directly. To begin with, there are no explicit criteria for rating APL idioms (much less APL code in general). Issues involving programming style are, of course, largely subjective and often controversial. Besides, there does not seem to be much need to judge idioms -- only to catalogue and use them.

Actually, answers to these questions may have importance beyond mere curiosity about which idioms are most popular. For instance, comparisons of idioms have implications for teaching APL: which APL idioms should students learn (and when)? Not all in the FinnAPL Library![1] And certainly not in the order given. Idiom ratings may also influence design of enhanced APL, perhaps in determining which functions to optimize or good candidates for implementation as future primitives.

This paper presents an experimental instrument -- called "APL IDIOM INVENTORY" -- which has been tested by a small number of APL programmers and instructors with some well-known APL idioms for try-out.

APL Idiom Inventory

This APL Idiom Inventory is comprised of ten scales which attempt to capture salient features of an idiom. One scale (LENGTH) is an objective measure; one (⎕IO) is a binary feature; the others necessarily involve some subjective judgments and may overlap somewhat, depending on individual interpretation. Ratings range from 0 to 10. A facsimile of the APL Idiom Inventory is shown on the next page.

---

* The term 'idiom' is used here due to its general acceptance in the APL community even though it is somewhat of a misnomer. (An idiom in English is an expression that, through usage, has come to be known as something other than its literal meaning. E.g., "Wait a second" and "Heads up!") In APL programming, the term 'phrase' may be more appropriate to denote a collection of symbols commonly recognized as a useful building block. Nevertheless, an APL idiom can be regarded as a phrase which has become known (often by a denotative name) for what it does, rather than by its strict, symbol-by-symbol interpretation.

APL IDIOM INVENTORY

Idiom: ______________________   LENGTH _____   ⎕IO ____

USEFULNESS

```
   0    1    2    3    4    5    6    7    8    9    10
   <----------------------+---------------------->
common                                          rare
```

EFFICIENCY

```
   0    1    2    3    4    5    6    7    8    9    10
   <----------------------+---------------------->
efficient                                    wasteful
```

GENERALITY

```
   0    1    2    3    4    5    6    7    8    9    10
   <----------------------+---------------------->
generalized                                  specific
```

CLARITY

```
   0    1    2    3    4    5    6    7    8    9    10
   <----------------------+---------------------->
reveals                                      conceals
```

SIMPLICITY

```
   0    1    2    3    4    5    6    7    8    9    10
   <----------------------+---------------------->
simple                                        complex
```

MEMORABILITY

```
   0    1    2    3    4    5    6    7    8    9    10
   <----------------------+---------------------->
rememberable                                 forgettable
```

INTERESTINGNESS

```
   0    1    2    3    4    5    6    7    8    9    10
   <----------------------+---------------------->
insightful                                      trite
```

ELEGANCE

```
   0    1    2    3    4    5    6    7    8    9    10
   <----------------------+---------------------->
elegant                                     inelegant
```

Explanation of Scales

LENGTH is the number of characters in the APL idiom.

⎕IO is the Index Origin assumed: 0 or 1 (⎕IO-dependent) or either (⎕IO-independent)

USEFULNESS is a measure of how commonly the idiom is found in applications. Please judge this on the basis of actual use of the idiom or an estimate of its use in the field.

EFFICIENCY is a combined measure of speed and space requirements for the idiom. Since this is system-dependent, please estimate over as many familiar implementations as possible.

GENERALITY is a measure of the idiom's ability to handle a wide range of cases -- including both numeric and character data types, arrays of higher rank, and special cases (such as scalars and empty arrays).

CLARITY is the extent to which the idiom reveals or conceals its underlying purpose. Some idioms are expressed directly; others distort their algorithms by using artificial, unusual, tricky, or bizarre coding techniques (albeit for efficiency in length or space or speed).

SIMPLICITY is a measure of how easy or hard it is to understand the idiom -- especially when learning it for the first time. Please take into account sophistication of primitive functions and operators involved as well as the complexity of expected data structures.

MEMORABILITY is a measure of how easy or hard it is to recognize (if reading) or to recall (if writing) the idiom. It may be remembered by rote or by reconstruction or whatever.

INTERESTINGNESS is a scale concocted to assess the extent to which the idiom is surprising, lends insights, or leads to fruitful interrelationships. One which doesn't have many connections is called "trite".

ELEGANCE is a very subjective scale which is left open to judge an idiom intuitively on aesthetic grounds.

A is any Array	V is a Vector
B is a Boolean	W is a vector
N is a Numeric scalar	M is a Matrix
L1 and L2 are line lables	

N.B. Whenever possible, please base ratings on comparisons with other idioms for accomplishing the same purpose. If no alternative is known, then compare to all familiar idioms.

Selected APL Idioms

For try-out, about a dozen idioms were chosen from the APL literature, guided by suggestions from APL experts. The idioms are listed below, along with colloquial names:

```
"Round"              ⌊0.5+A

"Sort"               V[⍋V]

"Unique"             ((V⍳V)=⍳⍴V)/V

"First 1"            <\B

"Same Boolean"       ≠/0 1∊B

"Scalarize"          ''⍴A

"Difference"         (¯1↓V)-1↓V

"Match"              ∨/M∧.=V

"Merge"              (V,W)[⍋⍒B]

"Identity Matrix"    (⍳N)∘.=⍳N

"Coerce to Matrix"   (¯2↑1 1,⍴A)⍴A

"If"                 →B/V

"Do N times"          I←0
                     L1: →(N<I←I+1)/L2
                      ...
                      →L1
                     L2: ...
```

While this is only a small sample of APL idioms, the limit of a (baker's) dozen was imposed here to ensure that the evaluators could finish their ratings within a reasonable amount of time (approximately a half hour).

A variety of different idioms were chosen, including idioms with 1, 2 and 3 arguments; arguments with 0, 1, 2 and unlimited ranks; some idioms for numerical processing, some for either data type, and some restricted to Booleans; some for iterative processes, and some for array-processing; some fully generalized, and some for specific arguments only; and, in general, idioms applicable to a wide range of disciplines.

Idioms omitted here include those which could be considered as specific application "tools", e.g., in text processing: `(-+/∧\⌽M=' ')⌽M` ("Right-Justify") and `(∨\V≠' ')/V` ("Delete Leading Blanks").

Also omitted were incomplete idioms such as `(∧⌿(¯1+⍳⍴W)⌽W∘.=V)/⍳⍴V` for string search (which fails for certain edge conditions) as well as long and complicated idioms, usually warranting embodiment in defined ("cover") functions, e.g., `I←(1-⍴W)↓V` ... `(V[I∘.+¯1+⍳⍴W]∧.=W)/I←(I←(I=1↑W)/⍳⍴I` also for string-searching.

Only ⎕IO-independent idioms were chosen here in order to be consistent, thereby excluding idioms like "Bar Graph" V∘.≥⍳⌈/V and "From" (scattered point selection) (,A)[1+(ρA)⊥⍉M-1] as well as "String Search" (both on previous page).

Idioms which cause side-effects (such as variable assignment) were avoided -- with the exception of "Do N times", for which I is expected to be localized.

Idioms which contain other idioms were also avoided, e.g., +/V∘.=((V⍳V)=⍳ρV)/V "Frequencies" (using "Unique"), if for no other reason than difficulty in separating out the influence of a sub-idiom.

Further, "obvious" idioms were not preferred -- that is, those whose meanings are no different from their direct literal translation, such as "Test for Empty" 0∊ρA and "Howmany Rows" 1↑ρM and "Last" V[ρV] (although it might be noteworthy to compare the last one with ¯1↑,A which is more general but results in a one-element vector).

In any case, it is assumed that each idiom is thought of as a unit, used frequently, and has a common name.

In particular, "Round" was chosen because it is usually thought of as rounding off N to the nearest integer rather than literally as "Floor of one half Plus N". The more general form for rounding off to P places -- (10*P)×⌊0.5+N×10*P -- was not chosen because its greater length begs for a defined function. (Both are mostly obviated by the Format primitive function, anyway.)

"Sort" was chosen in ascending order arbitrarily over V[⍒V] . Alternatively, V[⍋V×N] uses controlling variable N∊1 ¯1 and may be potentially twice as usable but is certainly less efficient; besides, Reverse can be used easily as a prefix to go from one ordering to the other.

"Unique" (or "Nub") is perhaps the most-often illustrated APL idiom and has been implemented as a primitive function in enhanced APL systems -- even though it fails for a scalar and doesn't generalize well. Compare it with (1 1⍉<\V∘.=V)/V . And compare ((V⍳V)=⍳ρV←⎕IO++⌿∧\M∨.≠⍉M)⌿M with (1 1⍉<\M∧.=⍉M)⌿M to remove duplicate rows of a matrix, and (1 1⍉<\1 3 3 2⍉A∧.=⍉A)⌿A for rank-3 arrays, etc.

"First-1" is one of many idiomatic uses of Scan -- one which seems to arise often in various applications to find the first position of a value in (rows of) an array. Alternative idioms are much more cumbersome or expensive or inelegant, e.g., 1=+\B . Also, beware that it works for any numeric array but with spurious meaning.

"Same Boolean" may not be used that often, but it has no less than ten alternative expressions for detecting either all 1s or all 0s ([2] p. 16). This idiom is not only shortest, but perhaps surprisingly simple. Further, it leads to the related problem of expressing an idiom for "Same Element" (which is done a different way): ∧/,A=1↑,A

"Scalarize is a good example of a simple idiom which the programmer may not want to think about each time, but rather just use. Indeed, '' instead of (⍳0) may be more economical but dissonant when A is numeric (fortunately, most interpreters are forgiving).

"Difference" has a mirror-image idiom in (1↓V)-¯1↓V ; this might cause pause in remembering the direction in which the differencing is to occur. It suggests the more primitive idiom "Shift" 0,¯1↓V and its relative "Restore" V-0,¯1↓V (for restoring the original vector from a Sum-scanned V).

"Match" seems to be a classic, works for either data type, and has other related forms, e.g., (M∧.=V)⍳1 for the index of the first matching row, and ∨/V∧.=M for matching by columns.

"Merge" requires three arguments and, consequently, is a good candidate for use as an idiom rather than as a defined function. It is, however, not a straightforward way to merge; compare with (B\V)[(~B)/⍳ρB]←W or (B\V)+(~B)\W (for numeric V and W).

"Identity Matrix" has applications beyond linear algebra, but is it more natural than alternatives (N,N)ρ(N+1)↑1 or (N,N)ρ1,Nρ0 ? Does symmetry help in its recall? (Is that why some people use redundant parens?) And does it lend any insight into how to generalize for diagonals of higher dimensional arrays?

"Coerce to Matrix" accepts an argument of any rank but returns only the first matrix for ranks greater than 2 and fails for special cases of empty rank 3 or greater arrays with non-empty rows and columns. The alternative idiom ((×/¯1↓ρA),¯1↑ρA)ρA doesn't lose any data but restructures the result and fails for a scalar.

"If" is often found as a defined function in utility workspaces and, of course, can be used as is for branching. Other similar idioms are: →N×⍳B (not allowing vector arguments and not ⎕IO independent) or →BρN or →B↑N and →B↓N .

"Do N times" is included as a single idiom even though it is written over several lines. It is one of several constructs for iterative programs and, for instance, might be compared with:

```
  I←1
 L1: ...
  →(N≥I←I+1)/L1
  ...
```

which is one line shorter but must "Do" at least one time. Another related construct is "If, Then, Else":

```
  →(~B)/L1
  ...
  →L2
 L1: ...
 L2: ...
```

Results

Results of the try-out of the above idioms are summarized below (with abbreviated scale names):

Scale / Idiom	LEN	⎕IO	USE	EFF	GEN	CLA	SIM	MEM	INT	ELG
"Round"	2.0	0.0	2.3	1.7	2.4	1.9	0.9	1.0	5.7	3.7
"Sort"	2.0	0.0	1.3	2.2	4.8	0.9	1.0	0.3	3.3	2.4
"Unique"	5.2	0.0	3.0	3.4	4.0	5.0	4.9	3.4	3.6	3.1
"First 1"	1.2	0.0	5.1	2.7	5.7	7.1	3.9	4.4	3.9	3.3
"Same Boolean"	2.8	0.0	7.6	3.6	4.8	5.6	4.0	5.7	5.9	4.2
"Scalarize"	1.6	0.0	1.9	0.9	0.5	3.4	2.1	2.1	5.3	5.2
"Difference"	4.0	0.0	3.5	3.0	5.7	2.5	3.4	4.5	4.8	5.2
"Match"	2.8	0.0	3.5	5.2	3.8	3.7	3.6	4.1	4.9	3.7
"Merge"	4.0	0.0	6.7	5.5	4.6	7.8	7.0	7.5	3.3	3.3
"Identity M"	3.6	0.0	3.7	6.3	5.8	2.2	2.6	1.1	3.6	3.1
"Coerce to M"	5.2	0.0	4.6	2.3	4.4	4.3	4.1	4.7	5.2	4.9
"If"	1.6	0.0	1.5	1.2	2.6	2.6	1.0	0.7	5.7	4.2
"Do N times"	10.0	0.0	0.6	2.4	2.9	2.6	2.9	1.6	6.7	6.1

Numbers are averages of ratings by all respondents (n = 10).

N.B. LENGTH has been adjusted to a scale of 0 to 10 by multiplying 10 times the idiom's length (number of characters) divided by the maximum length idiom given here (25).

N.B. ⎕IO is scored as either 0.0 for Index-Origin independent idioms or 10.0 for Index-Origin dependent idioms.

Discussion

The data suggest that the "Round" idiom is simple and easy to remember; "Sort" is useful, clear, simple, most memorable and comparatively elegant; "Unique" is ironically not unique in any respect; "First 1" is short but not very clear; "Same Boolean" seems to be rarely used; "Scalarize" is very efficient and general; "Merge" appears unclear, complex and hard to remember -- perhaps because it isn't used much; "Identity Matrix" may be somewhat inefficient and lacking generality but is easy to remember; "If" is useful, efficient, simple and rememberable; "Do N times" is very useful, but long and hardly interesting. In sum, "Sort" is possibly the overall best (best = minimum total distance from 0 on all scales).

There are some methodological issues which beg to be discussed. First, the reliability of the APL Idiom Inventory remains to be determined, as well as cross-correlations between scales. (This warrants a field test with large N and more idioms, of course.) Secondly, there is a confounding difficulty in comparing idioms; that is, whether an idiom must be compared strictly against idioms which accomplish the same purpose. (See note at bottom of Explanation of Scales.) Or is it reasonable to compare an idiom with other idioms -- outside its ecological niche? Are we not measuring 'survival of the fittest' anyway?* This must be resolved in order for the ratings to make sense.

----

* An idiom may have evolved from a basic need for a certain expression; it may have become popular because it got used often; it may minimize portability problems, say, by being ⎕IO-independent. When compared with other idioms, it may have proved to be the fittest because it was most efficient on certain computers and/or because it was the most concise in writing and/or because it was easiest to remember. Or, instructors may have taught it to other people simply because they liked it. For instance, consider ?1 as an alternative to ⎕IO (a rather idiosyncratic idiom, to be sure). You might prefer it because it is shorter. You might be repulsed by it because it uses a function totally unrelated to its purpose, resets ⎕RL, and is probably less efficient in execution. Or you might be pleasantly surprised because you never would have thought of it yourself. So, will you use it or not?

Other issues focus on the design of the instrument itself. For instance, USEFULNESS could be determined theoretically by a frequency count of all idioms in all existent APL code (not a welcome task!). And, EFFICIENCY ratings could be more accurate if it were known which primitive functions are optimized on particular machines. Also: How much overlap is there among scales such as CLARITY, SIMPLICITY, and MEMORABILITY? What is ELEGANCE anyway, and what does it correlate with? What should the relative weights of scales be (for determining a total rating)? After all, there may well be considerable differences between the author's descriptions of these scales and other people's interpretations.

For some programmers, utility is the only important feature of an APL idiom: it gets the job done. Other features, such as generality, may not be practically relevant: "If I never need it for higher rank arrays, why should I care?" Some people use APL idioms to think with; they bring to mind helpful chunks for solving problems or valuable identities for constructing proofs. (See [3] .) And, maybe some people use idioms without knowing it.

Eventually, an APL Idiom Inventory may help clarify fundamental questions about APL idioms. For instance, which are generally better -- idioms or "cover" (defined) functions? Defining a cover function does require investing extra effort to embody code and extra knowledge of function / group / workspace names (plus awareness of possible conflicts), but it may help understanding at least because the cover names are connotative; whereas an idiom itself must be retyped each time it is used and may be construed as just a collection of symbols to learn by rote. On the other hand, idioms may lead to better understanding because one must learn to recognize and use an idiom in context; whereas a cover function can be used blindly, without looking at its definition and perhaps forgetting special cases. So, which will prevail in the future? (Whose memory will we rely on -- ours or the computer's?)

Conclusion

In the meantime, we can now find out what the APL community thinks of various idioms -- by using an APL Idiom Inventory.

Readers are invited to evaluate the idioms here (plus any other favorite APL idioms). Please send ratings to the author along with suggestions for improving this instrument.

References

[1] "FinnAPL Idiom Library" (2nd Edition), Finnish APL Assoc., Helsinki, Finnland July 1982

[2] "The APL Idiom List", Perlis & Rugaber Computer Science Research Report #87, Yale Univ., New Haven, CT April 1977

[3] "APL Thinking: Examples", Eisenberg & Peelle, APL87 Conference Proceedings, APL Quote-Quad, (to appear) May 1987

[4] "Idioms and Problem Solving Techniques in APL2", A. Graham, APL86 Conference Proceedings, APL Quote-Quad, Vol. 16, No. 4, July 1986

Acknowledgements

Thanks to Professor Murray Eisenberg (University of Massachusetts), Dr. Mike Sutherland (Adaptive Data Systems, Inc.) and Jim Weigang (STSC, Inc.) for help in conceptualizing, testing, and refining this APL Idiom Inventory.