APL fonts are different (ver. 2)

Phil Chastney

Originally written for those who wish to tailor an existing font, these notes may also be of use to those curious to know why fonts need to be tailored specifically for APL.

The characteristics which concern us here are the encoding of the font (which binary value corresponds to a given character), and the glyphs (the physical display of a character, as seen by the user).

There are no rendering issues. The overstriking necessary with early impact printing systems is now entirely superfluous.

1. APL fonts are not what they were

The classic appearance of APL code is that seen on the IBM 2741 golfball terminal [1]. The characters A-Z are Courier light italic, while the rest of the character set is upright and of a similar weight. There is no lowercase a-z, the space being required for special symbols, but a second alphabet could be constructed using underscored uppercase letters.

Things changed with the arrival of the PC, with its lo-res raster screen, and the launch in 1992 of Windows 3.1, with its scalable TrueType fonts. Italic fonts do not render well on lo-res screens—staircasing is inevitable. Secondly, with impact printing, the ink from a lightweight face will bleed a little into the paper, to give a darker appearance, but no such effect occurs on screen. Most scalable APL fonts were therefore upright, and book weight or heavier, with some considerable variance in the choice of typeface for the alphabetic characters.

2. APL encodings differ from each other

Unicode 1.0 came out in October 1991, version 1.1 in June 1993 included some special APL characters, and by 1995 they were all in there.

Latin-1, APL's special characters, and much more, could now be made available within the same font file, at little extra effort. (Although no longer strictly necessary, preformed 3270-style underscored A-Z composites were sometimes included.)

At APL2000, in Berlin, agreement was reached on a mapping from APL to Unicode values — the APL Character Repertoire [2], which is hereafter referred to as “N3067 ACR”. Starting from Unicode’s requirement that every Unicode font include ASCII, the Repertoire also includes four arrows from 2190-21FF Arrows, sundry characters from blocks 2200-22FF Mathematical Operators and 2300-23FF Miscellaneous Technical, and one from 25A0-25FF Geometric Shapes.

That agreement never became a standard, either *de jure* or *de facto*, and we are left with a number of slightly different encodings for some of the special symbols, while retaining/redefining ASCII characters for others.

In addition to the required ASCII coverage, any font intended for the display of APL code should include glyphs against all the codepoints in N3067 ACR, plus some duplicates, as explained below.

3. APL’s glyphs differ from other Unicode fonts

3.1 ASCII

Some ASCII codepoints are ambiguous: U+002D *hyphen-minus*, for instance. Unicode retains the ambiguous character for continuity purposes, but also includes specific codepoints for the separate semantics of *hyphen* (U+2010) and *minus* (U+2212).

The ASCII asterisk is a raised character, distinct from U+204E LOW ASTERISK and U+2217 ASTERISK OPERATOR. The ASCII tilde, being a spacing variant of the diacritical marking U+0303 COMBINING TILDE, is also a raised character, distinct from U+223C TILDE OPERATOR.

The issue with all of these is vertical positioning, for horizontal alignment. A tidily drawn font will have a number of horizontal guidelines common to all glyphs: all European scripts sit on the baseline, for instance. Other guidelines are the caps height, x-height, the top of lowercase ascenders, and the bottom of (lowercase) descenders. Fences and delimiters may have different guidelines; superscript and subscript characters certainly do, while mathematical symbols are usually centred on a line somewhere close to half the caps height.

Because some implementations have opted to associate APL operators with ASCII codepoints, APL fonts need to do likewise. Identical vertically centred tildes are needed for U+007E and U+223C. The situation regarding asterisks is a little more complicated, as explained below.

Within the APL community, these discrepancies are concealed through the use of “tolerant input”, which allows, for example, both U+007E TILDE and U+223C TILDE OPERATOR to be keyed in, or cut and pasted in, and recognised by the implementation as denoting the same operation.

The discrepancies only really show when a specifically APL font is used for other purposes, and vice versa.

It takes a practised eye to detect the difference in vertical positioning of hyphen and minus. The difference in the lengths of the two glyphs, however, would surely be obvious to all, given a proportional font, but most APL fonts are still monospaced (possibly because it simplifies the interpreter's display problems).

3.2 diamonds and lozenges are different

So far as Unicode is concerned, a *diamond* is a square shape rotated 45° about its centre point. A diamond's diagonals are perpendicular, they mutually bisect and have the same length. A *lozenge* is a four-sided shape whose diagonals are perpendicular, and mutually bisect, but which are not necessarily of the same length (i.e, some lozenges are not diamonds.)

If we relax the condition that the diagonals mutually bisect, but just one diagonal bisects the other, the result is a more general *kite* shape.

The so-called "diamond" operator used in APL is a statement separator, or sequential operator. Being a later addition, there is no diamond in the APL\360 character set, and it was originally improvised by overstriking '<' and '>', or '∧' and '∨'. When scalable APL fonts appeared, the character in question was often shown as a lozenge.

Lozenge or diamond, for maximum compatibility, the chosen glyph needs to be shown against Unicode values U+22C4 (N3067 ACR recommended), U+25CA (vendor-specific) and U+25C7 (if you want to).

3.3 stars are not asterisks

Arrange *n* points equidistantly around the circumference of a circle, connect each point to its 2 nearest neighbours, and the result is an inscribed regular polygon.

Connect each point to the next point but one, and the result is an inscribed star. For *n*=5, the result is sometimes called an Arabic star; for *n*=6, the result is sometimes called the Star of David (U+2721).

From each point, skip 2 points, and connect to the 3rd point, and the result is another form of star. For *n*=8, the result is U+2738, one of Zapf's dingbats.

Another way of drawing a star is to place n points on the circumferences of 2 concentric circles, and join the dots. U+2735 is one example of a 6-point star which can be drawn by this method, while U+2734 is an example of an 8-point star.

In all these cases, the rays of the star are *kite* shapes. The rays of asterisks, on the other hand, are known as *petals*, for obvious reasons.

Codepoint U+22C6 is named STAR OPERATOR, with APL mentioned in the notes. The representative glyph is unmistakably an Arabic star, and is included in the list of stars of various magnitudes in UTR 25 [3]. U+2363, star-dieresis, and U+235F, circle-star, are both described as "APL FUNCTIONAL SYMBOLS", and they too show 5-point black stars.

One standardisation effort [4] decided, after some lively discussion, that the APL "star" is, in fact, an asterisk having 5, not 6, petals. Unicode contains a dozen or so asterisks, most of them in the Dingbats block, none of them with 5 petals.

All in all, N3067 ACR's recommendation of U+22C6 is probably the best codepoint for APL's 5-point asterisk, whose glyph will be centred on the same horizontal as other symbols. For maximum compatibility, the same glyph also needs to appear against codepoint U+002A (vendor specific) and U+2217 (just in case).

3.4 delta and del are not triangles

Delta and Del are letters: allowing a little for overshoot, they sit on the baseline and extend up to caps height; they are not vertically centred like + and ×. The APL Character Repertoire recommends codepoints U+2206 and U+2207, and this is where things get a little strange.

The entries for these characters read (in part):

2206	INCREMENT
	= Laplace operator
	→ 25B3 white up-pointing triangle
2207	NABLA
	= gradient, del
	• used for Laplacian operator (written with superscript 2)
	→ 25BD white down-pointing triangle

The Nabla is certainly used for the gradient function. According to Wikipedia (http://en.wikipedia.org/wiki/Gradient):

> The gradient (or gradient vector field) of a scalar function *f*(*x*) with respect to a vector variable $x = (x_1, \ldots, x_n)$ is denoted by ∇f [...] where ∇ (the nabla symbol) denotes the vector differential operator, del. The notation $\mathrm{grad}(f)$ is also used for the gradient. The gradient of *f* is defined to be the vector field whose components are the partial derivatives of *f*. That is:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right).$$

Wikipedia again (http://en.wikipedia.org/wiki/Laplace_operator):

> In mathematics and physics, the Laplace operator or Laplacian, [is] denoted by Δ or ∇^2.
>
> The Laplace operator is a second order differential operator in the *n*-dimensional Euclidean space, defined as the divergence ($\nabla\cdot$) of the gradient (∇f). Thus if *f* is a twice-differentiable real-valued function, then the Laplacian of *f* is defined by
>
> $$\Delta f = \nabla^2 f = \nabla \cdot \nabla f.$$

Quoting selectively from Springer on the same matter, (http://eom.springer.de/H/h046240.htm) we find:

> **Hamilton operator,**
> *nabla operator*, ∇*-operator*, *Hamiltonian*
>
> The application of the Hamilton operator to a scalar function f, which is understood as multiplication of the "vector" ∇ by the scalar $f(x)$, yields the gradient of f:
>
> $$\text{grad } f = \nabla f = \sum_{j=1}^{n} \mathbf{e}_j \frac{\partial f}{\partial x_j}.$$
>
> The scalar square of the Hamilton operator yields the Laplace operator:
>
> $$\Delta = \nabla . \nabla = \sum_{j=1}^{n} \frac{\partial^2}{\partial x_j^2}.$$

We seem to have come a long way from font issues, but the point of this digression is to establish that nabla, the gradient operator, grad, del and the Hamilton operator are different signs and names for the same thing [5].

Now this is the odd bit: the notes for U+2206 cross-reference U+25BD, whose entry reads (in part):

```
        25BD  WHITE DOWN-POINTING TRIANGLE
              = Hamilton operator
```

Cross references, according to the Unicode Standard (p566 of version 5.0), indicate Explicit Inequality. In effect, the standard is specifying one character for nabla, the gradient operator, grad and del – and a different one for the Hamilton operator, which makes about as much sense as having one symbol for *times*, and an explicitly different one for *multiply*.

This is clearly an error in the Unicode standard, so the note "= Hamilton operator", specifying an alternative name for U+25BD, should be ignored. And, consequently, the glyph appearing against Unicode value 2207 should not be used for Unicode value 25BD (which is, after all, a Geometric Shape, and therefore vertically centred).

3.5 NAND and NOR are different from NAND and NOR

It is often claimed that characters are admitted to the Unicode standard on the basis of distinguishable semantics, their actual appearance being decided outside the standard.

This may be true for natural languages, but not so for formal notations, where shape is all, and meaning may change with time and context. U+002B PLUS SIGN, for instance, may variously denote addition, logical "or" or string concatenation, while the uses of "Σ" are even more varied.

Even so, it comes as some surprise to find that NAND and NOR have each been included twice, with identical semantics but different glyphs. The character names are:

22BC	NAND
22BD	NOR
2372	APL FUNCTIONAL SYMBOL UP CARET TILDE
2371	APL FUNCTIONAL SYMBOL DOWN CARET TILDE

The first pair are defined by their semantics, the second pair by their shapes [6]. U+22BC and U+2372 have identical semantics, but different representative glyphs (the former being shown with a macron above, the latter overstruck with a mid tilde); U+22BD and U+2371 likewise.

An APL font will need glyphs for U+2372 and U+2371, using tilde above, or overstriking with a mid tilde, according to taste. U+22BC and U+22BD could also be included, but they are not essential.

3.6 | / \

The addition of U+29F5 to the Unicode standard means that each of these characters – the vertical stroke, the forward sloping version and the backward sloping version – is now defined twice:

U+002F	SOLIDUS	U+2215	DIVISION SLASH
U+005C	REVERSE SOLIDUS	U+29F5	REVERSE SOLIDUS OPERATOR
U+007C	VERTICAL LINE	U+2223	DIVIDES

The notes describe U+2215 DIVISION SLASH as the "generic division operator", U+29F5 is named as an OPERATOR, while the name "DIVIDES" for U+2223 strongly suggests that, it too, is an operator (more specifically, a predicate).

N3067 ACR recommends U+2223 DIVIDES for the APL modulus symbol. This has merit: as an operator U+2223 (= APL stile, according to the notes) will therefore line up with +, -, ×, ÷, etc. On the other hand, U+007C VERTICAL LINE, which some APL implementations prefer, is (in the terminology of UTR 25) a "fencepost"[1].

And here we have the reason for this apparent duplication: the trio on the right of the above table certainly denote operators, while the trio on the left presumably denote "fenceposts".

Delimiters and fences sometimes follow the same guidelines as the alphabetic characters, but not always. Sometimes fences and delimiters extend the full fg-height, from the top of the ascenders to the bottom of the descenders, with a centre line all of their own.

To avoid visual differences between implementations, APL fonts need to provide vertically centred glyphs for both U+2223 and U+007C.

For compression and reduction, on the other hand, N3067 ACR recommends U+002F SOLIDUS (= slash, according to the notes) to denote compression and reduction. The slash is (implicitly) a fence, and linguists, for instance, use square brackets to enclose phonetic sequences, and slashes to indicate the start and end of (i.e, to delimit) a sequence of phonemes. In the circumstances, the best we can do is to provide visually identical slashes for U+002F SOLIDUS and U+2215 DIVISION SLASH.

It may be a very long time before any APL implementor decides to use U+29F5 for expansion and scansion but, given the above, there may be value in providing identical vertically centred glyphs for both U+005C REVERSE SOLIDUS (= backslash) and U+29F5.

4. scalable APL fonts are not what they were

circles are different

APL uses two sizes of circle. N3067 ACR recommends U+25CB WHITE CIRCLE for the "circle", U+2218 RING OPERATOR for the "jot", and U+233E APL FUNCTIONAL SYMBOL CIRCLE JOT for the two combined.

Unicode has lots of circles, black and white, of various sizes and UTR 25 has defined an ordering on the sizes of these circles. Although Unicode claims not to be prescriptive regarding actual glyphs, the uses and conditions outlined in UTR 25 indirectly specify the sizes of the actual glyphs quite closely.

The net effect is that U+25CB is now rather larger than the traditional glyph, and U+2218 rather smaller, while U+233E is unchanged.

[1] Also called "fences" and/or "delimiters" – terminology is not standardised.

A font produced to UTR 25’s recommendations will not display APL circles and jots in a pleasing manner, while the fact that U+233E is unchanged introduces an unwanted inconsistency. Fonts for APL should therefore ignore UTR 25’s recommendations on shapes and sizes [7].

5. some clarification

N3067 ACR includes three Notes on items that were not 100% clear at the time of the APL2000 conference in Berlin.

5.1 Note 1: tacks

A left-pointing arrow will have its point on the left, and such a character is usually known as a “left arrow”. Unicode includes quite a selection of sharp objects—arrows, harpoons, tacks and daggers, for a start—and in all cases, the character is named according to the position of the point.

Well, actually, in *most* cases.

The first set of tacks were introduced as mathematical, rather than specifically APL, symbols, and the naming convention is consistent with that used for arrows:

22A3	LEFT TACK
22A2	RIGHT TACK
22A5	UP TACK
22A4	DOWN TACK

Compare and contrast with the next set of symbols:

2355	APL FUNCTIONAL SYMBOL UP TACK JOT = down tack jot
2361	APL FUNCTIONAL SYMBOL UP TACK DIAERISIS = down tack dieresis
2351	APL FUNCTIONAL SYMBOL UP TACK OVERBAR = down tack overbar
234A	APL FUNCTIONAL SYMBOL DOWN TACK UNDERBAR = up tack underbar
234E	APL FUNCTIONAL SYMBOL DOWN TACK JOT = up tack jot

The character definition for U+234A also carries the note:

- preferred naming for APL tack symbols now follows the London convention in ISO/IEC 13751:2000 (APL Extended)

There was, within the APL community, some inconsistency in the naming of tacks: the “London” convention named each tack according to the position of the point, while the so-called "Bosworth" convention named each tack according to the position of the thumb pushing it.

Unicode's commitment never to rename a character means that period of indecision is frozen forever in the inconsistent naming of these 5 symbols.

5.2 Note 2: epsilon

To quote N3067 ACR, "This symbol is one for which a truly good choice does not exist; the symbol chosen seems to match what is used in most systems better than available alternative".

The recommendation given was U+220A, which time has shown to have been the right choice. Lest any fontmonger be tempted to duplicate the glyph against other codepoints, they should take note that U+2209 extends to the full caps height, while U+220A is cross-referenced to (and is therefore deemed distinct from) "03F5 Greek lunate epsilon symbol".

5.3 Note 3: quad

In 2000, the recommendation for the APL Quad was U+25AF WHITE VERTICAL RECTANGLE, with the comment that "[the] square chosen was thought likely to cause the least problems, although admittedly it is not as much wider than the Squish Quad symbol as might be desirable".

U+2395 APL FUNCTIONAL SYMBOL QUAD was subsequently added to the Unicode standard (reluctantly, according to some reports). This is really good news, because it has protected the Quad symbol from the deleterious effects that UTR 25 had on Circle and Jot.

The Quad symbol can, and should, be treated as a letter, sitting on the baseline, extending to caps height. For maximum compatibility, it should appear against codepoint 2395 certainly, and 25AF probably.

U+2337 APL FUNCTIONAL SYMBOL SQUISH QUAD is unaffected by the introduction of U+2395.

5.4 comparisons

The APL\360 symbols for weak inequalities use horizontal bars. N3067 ACR recommends codepoints U+2264 LESS-THAN OR EQUAL TO and U+2265 GREATER-THAN OR EQUAL TO, which also have horizontal bars.

Unicode provides, in addition, U+2A7D and U+27AE, which have sloping bars. The difference in appearance is crucial, and this is a case where duplication of glyphs is definitely NOT required.

6. so they’re different – does it matter?

No, not a lot.

The objective is to define a font which can display all vendors’ code equally attractively.

APL can, of course, be rendered using Arial Unicode MS, or any other general purpose font suitably equipped with the necessary symbols. The display will be unambiguously readable, but may be visually rather inconsistent and unattractive.

The appearance can be improved by applying a simple, implementation specific, translate table each time the interpreter writes to a display device, but this approach can get messy.

As coverage of the more esoteric corners of Unicode improves, there may be other fonts with a similarly adequate character set, but if vendors continue to use U+002A ASTERISK as an operator, and users want to see a vertically centred asterisk for that operator, there will continue to be a need for a font specifically tailored for APL display.

In order to display all vendors’ code, there will necessarily be some duplication within this font, but installing one font specifically for all implementations is better than different fonts for different vendors.

The duplication and repositioning of glyphs will, to some extent, make an APL font unsuitable for general purpose display.

This in turn means that vendors will have to continue to supply and install special fonts, and/or provide the user with some means of specifying the font(s) to be used in the APL environment.

The only people likely to feel badly affected by this need for different fonts are those doing text processing in APL (i.e, something more advanced that assembling prompts).

Text processing often requires a large character set, so text processing in APL may mean having to use a font not specifically designed for APL. This is not a serious issue. Industrial strength text processing is usually done in scripting languages with regular expression facilities built in.

So, no, it does not matter – it’s a mild irritation, that’s all.

7. coda

This started out as a list of things to do. Verifying the statements on that list, and checking it for completeness, led to a need for a better structure.

Once the information was structured, it seemed like a good idea to save it, and hence to maintain it.

Errors and omissions will be corrected as they are found. Other material will change slowly, and this document will be updated when the change is discovered.

Presentation could be improved with a few illustrations, and that may happen eventually.

In the meantime, APL fonts are different.
And that’s the way it is likely to stay.

[1] http://bitsavers.org/pdf/ibm/apl/APL_360_Users_Manual_Aug68.pdf , page 1.3

[2] the APL Character Repertoire may be seen at http://www.dkuug.dk/jtc1/sc22/open/n3067.pdf

[3] UTR 25 is "Unicode Technical Report #25: Unicode Support for Mathematics", and version 11 may be found at http://www.unicode.org/reports/tr25/

Unicode Technical Report 25, Unicode for Mathematics, first appeared in 2001. (The name change came later, and was retro-actively applied to the archived copies of the earlier versions.) Discussion of the relative sizes of Geometric Shapes was qualified with [ED: TBD: summary picture], until Rev. 6 in 2003.

[4] The same standardisation effort also argued over whether the dollar sign was a part of the APL character set. Since any Unicode font must include ASCII, and the dollar sign, at U+0024, is a part of the ASCII block, every Unicode font, including APL fonts, must include the dollar sign.

[5] It is perhaps worth remarking that, although Hamilton used the Del symbol, he did not use it to denote the entity we now know as the Hamilton operator.

[6] This raises a couple of interesting questions:
— if a character is defined by its semantics, how far may an actual glyph vary from the representative glyph?
— if a character is defined by its shape, how far may its usage vary from that expected when the character was admitted to the standard?

Nand and Nor are, individually, logically complete, i.e, Nand alone is capable of expressing all the logical functions, and or Nor likewise. This was first *discovered* by C.S.Pierce, who (among other notations) used an up arrow for Nand, and a down arrow for Nor.

In the academic rat race, first publication date is the crucial measure, and in 1932 Sheffer made and *published* his discovery that Nor is logically complete (http://undaimonia.blogspot.com/2009/02/sheffer-stroke-problem.html contains a link to the original paper). Wittgenstein's N-operator is Nor, extended to arbitrarily many propositions (http://www.4vbc.com/wittgensteinsvariable.htm).

Sheffer used a vertical bar, now known as the "Sheffer stroke", to denote Nor, but the semantics of that symbol shifted over time, and it is now almost universally used to denote Nand. As an example, we reproduce here, in its entirety, the entry at http://mathworld.wolfram.com/ShefferStroke.html :

SEE: NAND

The Nor function is also known as Quine's "dagger" and von Neumann's "chalice". (See http://www.amazon.com/Logic-Language-Meaning-Introduction/dp/0226280853/ref=sr_1_2?ie=UTF8&s=books&qid=1238410285&sr=1-2 . Although the reproduction of page 57 at Google Books appears to use the Yen symbol, this may be an encoding problem.)

In summary, even when the semantics are defined clearly, popular usage may cause them change over time, so the answer to both interesting questions appears to be: choose the shape, regardless of its Unicode value, and use it to denote whatever semantics you want. Of course, life will be easier in the long run if the chosen shape has an appropriate set of properties.

7 The way UTR 25 defined the sizes of shapes had a similar effect on Z notation (see http://www.mcs.vuw.ac.nz/courses/COMP426/2006T1/2002/Documents/Z-standard.ps), most notably on the symbol used for composition of functions.