One Past the End

This starts off sounding complicated, but is actually very easy, especially towards the end. Trust me.

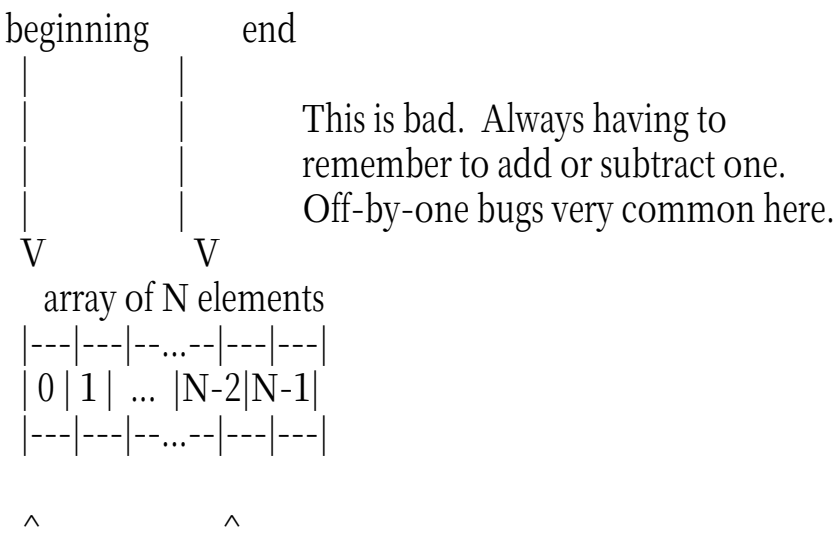
Beginners usually have a little trouble understand the whole 'past-the-end' thing, until they remember their early algebra classes (see, they *told* you that stuff would come in handy!) and the concept of half-open ranges.

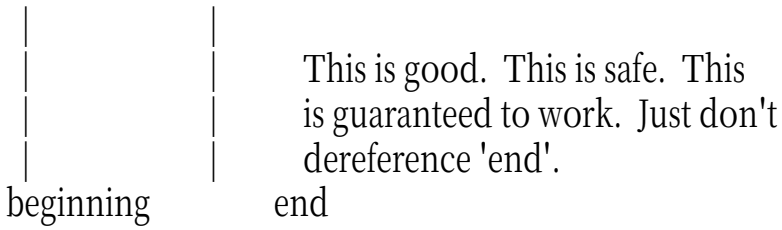
First, some history, and a reminder of some of the funkier rules in C and C++ for builtin arrays. The following rules have always been true for both languages:

1. You can point anywhere in the array, *or to the first element past the end of the array*. A pointer that points to one past the end of the array is guaranteed to be as unique as a pointer to somewhere inside the array, so that you can compare such pointers safely.
2. You can only dereference a pointer that points into an array. If your array pointer points outside the array -- even to just one past the end -- and you dereference it, Bad Things happen.
3. Strictly speaking, simply pointing anywhere else invokes undefined behavior. Most programs won't puke until such a pointer is actually dereferenced, but the standards leave that up to the platform.

The reason this past-the-end addressing was allowed is to make it easy to write a loop to go over an entire array, e.g., while (*d++ = *s++);.

So, when you think of two pointers delimiting an array, don't think of them as indexing 0 through n-1. Think of them as *boundary markers*:





See? Everything between the boundary markers is part of the array. Simple.

Now think back to your junior-high school algebra course, when you were learning how to draw graphs. Remember that a graph terminating with a solid dot meant, "Everything up through this point," and a graph terminating with an open dot meant, "Everything up to, but not including, this point," respectively called closed and open ranges? Remember how closed ranges were written with brackets, *[a,b]*, and open ranges were written with parentheses, *(a,b)*?

The boundary markers for arrays describe a *half-open range*, starting with (and including) the first element, and ending with (but not including) the last element: *[beginning,end)*. See, I told you it would be simple in the end.

Iterators, and everything working with iterators, follows this same time-honored tradition. A container's begin () method returns an iterator referring to the first element, and its end() method returns a past-the-end iterator, which is guaranteed to be unique and comparable against any other iterator pointing into the middle of the container.

Container constructors, container methods, and algorithms, all take pairs of iterators describing a range of values on which to operate. All of these ranges are half-open ranges, so you pass the beginning iterator as the starting parameter, and the one-past-the-end iterator as the finishing parameter.

This generalizes very well. You can operate on sub-ranges quite easily this way; functions accepting a *[first, last)* range don't know or care whether they are the boundaries of an entire {array, sequence, container, whatever}, or whether they only enclose a few elements from the center. This approach also makes zero-length sequences very simple to recognize: if the two endpoints compare equal, then the {array, sequence, container, whatever} is empty.

Just don't dereference end().
