M is for methodology —the programming sort.
It's not caught on big, but don't sell it short.
A sonnet's too long to capture its worth,
But a Haiku will work, if it's absent of mirth.

**Programming methodology**
What economy,
elegance, simplicity,
beauty, poetry

Cornell was heavily involved in formal programming methodology almost from the start, e.g. with the first text to take correctness issues seriously (Conway & Gries, 1973), automated proof checking (Constable, 1974 onward), an award-winning paper on proving parallel programs correct (Owicki & Gries, 1975), a text on the science of programming (Gries, 1981), fault tolerance (Schneider & students, 1980s), a discrete math text emphasizing calculational logic (Gries & Schneider, 1990s), and a comprehensive text on concurrent programming (Schneider, 1997).

Unfortunately, methodological issues have not been integrated into the undergrad curriculum as expected. In fact, the word "invariant" doesn't even appear in most intro-to-programming texts. The reason, I think, is that teaching a "science" of programming requires teaching a skill rather than simply facts, and instructors themselves have not been willing to learn the skills. It is a pity, because with suitable education in programming methodology, later courses in data structures, algorithms, etc. become more efficient and effective. Moreover, students are missing the sheer joy and intellectual fun that comes from watching a beautiful algorithm emerge from following a methodology. The elegance, the simplicity, the poetry in programming are missing.

```
/** Assume virtual elements b[-1] = -infinity
    and b.[b.length] = +infinity.
    Return a value i that satisfies
    R: b[i] <= x < b[i+1] */
public static int binarySearch(int[] b, int x)
int i= -1; int j= b.length;
// inv: b[i] <= x < b[j] && -1 <= i < j <= b.length
while (j != i+1) {
    int e= (i+j)/2;
    if (b[e] <= x) i= e;
    else j= e;
}
return i;
```

*"…back in 1986, I wasn't very impressed, and I didn't understand why I had to learn that stuff. … But with the passing of the years, I have found that … [what] you imparted to me —against my will— was the most valuable thing I learned at Cornell. You taught us how to do proofs of correctness of programs in the languages and environments, at the level that we would need to do them in the real world, at a level of detail that gave real assurance of correctness of code."*

*~ A Cornell PhD alumnus, in 2005*

N's for NA —but it's not "Not Applicable".
Its practical use is indeed undeniable.
What is NA, a new student asks.
We turn to Trefethen, who in the field basks.
Alg'rithms, he says, the study of which
Solve problems of math in th' continuous niche

Numerical analysis, as Nick Trefethen, our former colleague, will tell you, is the study of algorithms for problems of continuous mathematics —it is not just about rounding errors, accuracy, and approximation.

We are proud to have been a substantial player in the field of NA, or scientific computing as it is now called, right from the start. Jim Bunch (co-author of Linpack), Jorge More, John Dennis, Tom Coleman (now Dean at Waterloo), and Trefethen (now at Oxford University) all spent substantial time here.

The NA group is led by Charlie van Loan, whose coauthored book *Matrix Computations* is one of the most widely cited text in the computing and math sciences, and Steve Vavasis, whose automatic mesh generator is one of the most important software tools in solving boundary problems over irregular domains. Paul Chew's work on mesh generation needs mentioning too.

The Cornell environment has helped NA flourish here, with strong ties to Math and the Center for Applied Mathematics. Interdisciplinary work is stronger than ever before. For example, there's Uri Keich's work on BLASTn for matches in DNA sequences, and Keshav Pingali's advances in grid computing has contributed to the success of the work of Civil Engineer Tony Ingraffia and Vavasis on crack propagation in airplanes. And, as Van Loan will tell you, PageRank is an eigenvector computation!