

IMPROVING SOFTWARE PROGRAMMING

I can't help thinking that, for the most part, the author of "VIM: Taming Software with Hardware" (M. Halpern, Oct. 2003, pp. 21-25) is trying to solve a nonissue with a nonsolution.

First, I would like to see a specific memory-accessing model that allows me to access my data. This model must allow a programmer to differentiate elements within a collection and access any particular element reliably by a defined index of some sort. Then we can get into discussing the merits and drawbacks of this model.

Simply saying "virtually infinite memory" doesn't cut it because I need read/write access, not write-only access, to each identifiable subelement of my data. Arrays, therefore, are not expedients we resort to to conserve memory but a logical representation of the data plus a method to uniquely reference each element in that collection.

Second, I would like to see a suggested replacement for loops. The operations I need are:

- Perform the desired operation over a certain subset of my data—which I cannot calculate before that exact moment at runtime, and that I may not even be able to calculate during the operations. Current statements: for ($i=a.begin(); i!=a.end(); i++$) { operations; }; and while (! flag) { data operations; }.
- Repeatedly extract elements from an I/O device (file, socket, and so on) and process them until an EOT/EOF condition occurs: while (! eof(instream)) { data operations; }.

Third, programming is much different today than it was in 1980. Today, I can write a program that uses key-based array accesses to a "virtually infinite" collection. At work, I use MSVC,



so I use its built-in set, map, and list classes, but equivalent classes should either exist in most modern programming languages or be easily written from any handy book on data structures and operations.

In addition, most languages can dynamically create arrays of whatever size you need (up to available memory) using the new operator or the malloc() or an equivalent function. Programmers are no longer "forced" to allocate fixed-length arrays for buffer space; instead, they can fit the memory required to the problem at hand. Thus, the argument of array accesses being "bad" is mostly moot from this perspective—if you don't like fixed-length arrays, don't use them.

Thus,

- Throwing computer resources at the problem can ameliorate, but not eliminate, the effects of poor programming.
- We already have the tools we need to "solve" this "problem."

*Michael J. Lewchuk
Edmonton, Alberta
michael.lewchuk@shaw.ca*

Mark Halpern responds:

Michael Lewchuk writes that VIM is a nonsolution to a nonproblem. My first reaction is, hey, sounds like a plan! But the serious answer is that his objections are based on a misconception: VIM is not intended to change the way

a programmer expresses his intentions; it is, in fact, directed to the way his intentions are carried out. It's the object program, not the source program, that VIM aims to change radically, and it's loops, not arrays, that I referred to as "simply expedients we resort to to conserve memory."

Using VIM, programmers still create arrays, and they refer to their elements by array names and subscripts—the "memory-accessing model" remains unchanged. What can differ significantly is the way VIM represents arrays internally.

As for dynamic arrays, a VIM system, like a conventional one, can allocate space only when requirements become known. The difference is that VIM can allocate *memory*—it does not need to juggle pages and interrupts and all the other nonproductive constructs of a memory-bound system. Lewchuk says that "...most languages can dynamically create arrays of whatever size you need (up to available memory)." They sure can; they can even create—conceptually—arrays that far exceed available memory. Such creation is carefree, but storing real data is another matter.

Finally, I made no claim that VIM would reduce the incidence of poor programming (although it would, at least, ensure that programmers would not botch memory management). And, while we may have the tools to resolve what Lewchuk refers to as a "nonproblem," VIM addresses a real problem—unless the burden of masses of unnecessary memory-management software is not considered a problem.
*Mark Halpern
markhalpern@iname.com*

We welcome your letters. Send them to computer@computer.org. Letters are subject to editing for style, clarity, and length.