Subject: Re: saving variables between calls to a procedure?
Posted by JD Smith on Thu, 01 Aug 2002 01:38:51 GMT
View Forum Message <> Reply to Message

On Wed, 31 Jul 2002 16:12:46 -0700, Mark Hadfield wrote:

> "David Fanning" <david@dfanning.com> wrote in message
> news:MPG.17b21214ff7d9b35989944@news.frii.com...
>
>> Paul van Delst (paul.vandelst@noaa.gov) writes:
>>
>>>> pro define,ptr
>>>> *ptr=[*ptr,10]
>>>> end
>>>
>>> Hmm. That seems like an extremely dangerous thing to do - couldn't you
>>> clobber something by concatenating like that? If IDL is smart enough
>>> to recognise that the next bit of memory may be used by something else
>>> it then seems that you would end up with a non-contiguous data
>>> structure (in the figurative).
>>
>> This doesn't seem dangerous to me (perhaps because I use the construct
>> all the time). It seems like one of those wonderful things IDL
>> occasionally does that makes you think to yourself "Now, by God, that's
>> how software *ought* to work!"
>>
>> In any case, it works, over and over and over. And it never occurred to
>> me that non-contiguous data storage could be involved, even remotely.
>
> Extending an array a with the a = [a,b] syntax doesn't create a
> non-contiguous data structure. What is does is create a new array a,
> insert the elements from the existing a and b into it, then delete the
> old a. This is fine for small arrays but it slows down on large arrays
> because of all the memory allocation & deallocation. It is *very* bad
> practice to create a large array by growing it one element at a time.
>
> What do I mean by "large" in this context. I don't know, a few thousand
> I guess. Here is an exercise for the reader: time the following code for
> various values of n:
>
> a = [0]
> for i=1,n-1 do a = [a,0]

I just realized a factor of two increase in speed in a case of very
repetitive use of the a=[a,b] mechanism with vectors of length 10 or less!
So yes, a=[a,b] concatenation is very slow when compared to setting aside
the memory to begin with, and just incrementing an index. Unfortunately,
you don't always know how long your array will eventually be (luckily, in

my problem I had a reasonable upper bound on the number). Decent languages provide dynamic arrays which pre-allocate memory in increasing blocks, and thus avoid the overhead of malloc() on every round, not to mention maintaining flexible array endpoints to keep push and shift operations O(1). It's just absolutely ridiculous to copy the array everytime you'd like to add something to it.

In "fast" languages like C without advanced memory handling ala dynamic arrays, this type of problem is usually solved by a linked list, for which addition, insertion, and deletion are all O(1) operations. Unfortunately, the high overhead of IDL's pointers (which are nothing like machine-level C pointers) limit the utility of this method too (although I guess I should test that statement).

Anyway, the lesson is, beware of a=[a,b], which is convenient, but costly.

JD
