
Subject: Re: dynamic array workaround?

Posted by [David Fanning](#) on Mon, 15 Aug 2005 00:55:21 GMT

[View Forum Message](#) <> [Reply to Message](#)

Jeff N. writes:

> I haven't seen anything in the help files that would lead me to believe
> that IDL has a way to do something like the following:
>
> Say I want to loop through a few lists of numbers, and create one big
> array of all numbers in the lists that meet some criteria. Since I
> don't know beforehand how many numbers will meet the criteria, I don't
> know how many elements that one big array will need.
>
> If I were writing code in a language like PERL maybe, I'd use a dynamic
> array, but I don't think that IDL supports those. Any idea as to how I
> could get around this? What I've been doing is to just initialize a
> scalar, then assign new elements to it in the loops, then just removing
> the 1st element when I'm done looping. I guess I could also loop
> twice, with the first round of loops serving to find out how many array
> elements I'd end up needing. Both approaches seem inefficient to me.
> What else could I try? I had a sneaky suspicion that the TEMPORARY
> function was what I needed, but somehow that doesn't quite look right
> either.

I'd probably use a pointer to do this:

```
IF Ptr_Valid(ptr) THEN ptr = Ptr_New([mylist]) ELSE $  
  *ptr = [Temporary(*ptr), mynewList]
```

```
finalList = Temporary(*ptr)
```

If your lists contain integers, you can probably use the HISTOGRAM function to do the concatenation. See the Histogram tutorial and the article on Set Operations on Arrays:

http://www.dfanning.com/tips/histogram_tutorial.html

http://www.dfanning.com/tips/set_operations.html

Cheers,

David

--

David Fanning, Ph.D.

Fanning Software Consulting, Inc.

Coyote's Guide to IDL Programming: <http://www.dfanning.com/>

Subject: Re: dynamic array workaround?

Posted by [Robert Moss](#) on Mon, 15 Aug 2005 02:09:27 GMT

[View Forum Message](#) <> [Reply to Message](#)

David Fanning wrote:

```
>
>
> I'd probably use a pointer to do this:
>
> IF Ptr_Valid(ptr) THEN ptr = Ptr_New([mylist]) ELSE $
>   *ptr = [Temporary(*ptr), mynewList]
>
> finalList = Temporary(*ptr)
>
>
```

You should be careful if your number of concatenations become large. Using the method David suggests is fine for small n (I use it frequently), but if (say) you wanted to do it 90,000 times, the overhead becomes prohibitive. For example, the test program at the end of this message produced the following results:

```
% Compiled module: DELETEME.
IDL> deleteme
method 1 time elapsed = 17.2340002060
method 2 time elapsed = 0.0159997940
```

Method 1 was using David's method (again, perfectly fine for small n), and method 2 simply guesses at a large number of elements and counts the ones being used. This is much, much faster. If you really have no idea how big to guess, you could use CATCH to extend the array by another large chunk if it fills up.

Robert Moss, Ph.D.

```
pro deleteme
  compile_opt idl2

  t0 = systime( /sec )
  for i = 0, 90000 do begin
    IF ~Ptr_Valid(ptr) THEN ptr = Ptr_New([ i ]) ELSE $
      *ptr = [ Temporary(*ptr), i ]
  endfor
  finalList = Temporary(*ptr)

  print, "method 1 time elapsed = ", systime( /sec ) - t0, format =
```

```
'(% "%s",D13.10)'
  PTR_FREE, ptr

t1 = systime( /sec )

finalList = intarr( 200000, /nozero )
count = 0
for i = 0, 90000 do begin
  finalList[ i ] = i
  count++
endfor
finalList = finalList[ 0:count-1 ]
print, "method 2 time elapsed = ", systime( /sec ) - t1, format =
'(% "%s",D13.10)'

end
```
