Subject: Re: How to use pointers instead of arrays
Posted by JD Smith on Mon, 09 Dec 2002 17:51:49 GMT
View Forum Message <> Reply to Message

On Mon, 09 Dec 2002 07:56:44 -0700, David Fanning wrote:

> Murat Maga (maga@mail.utexas.edu) writes:
>
>> The question is I need a 2 dimensional array to which new elements
>> constantly appended during the execution. Currently I create a
>> duplicate temp array, create a new one with right size, and finally
>> transfer the values from the temp one to the actual one. I know people
>> use pointers for things like this, but I never had an example. Can
>> somebody post me a simple example? Do the things speed up if I use
>> pointers?
>
> People probably do use pointers for these sorts of things, but if they
> do it doesn't solve their problem. :-)
>
> The real problem is one of memory management. And continually creating
> and recreating arrays is bad business no matter how you do it, even with
> pointers.
>
> What you want to do is allocate memory in big enough "chunks" that it is
> efficient and meets the needs of your program. For example, if the
> number of "things" you are going to put into your array ranges from 10
> to 1000, then you might allocate memory to your array in chunks of 100.
>
> In practice, this means that you have some kind of counter to tell you
> where you are in your array. If the counter gets above the "chunk" size,
> you allocate more memory to the array:
>
>   array[counter] = value
>   counter = counter + 1
>   IF counter MOD 100 EQ 0 THEN array = [Temporary(array), Findgen(100)]
>

Another technique, useful when you really have no idea how much space
you'll need in the end, is to start with some reasonable increment,
and then double it each time you extend the array. E.g. 100, 300,
700, 1500, etc.

He might also be thinking of linked lists, or equivalent structures in
which creating the additional memory and copying is unnecessary, but
actually accessing the data requires you to traverse some
pointer-linked memory structure. This is difficult in IDL, and will
probably be slower overall.

<pieintheskydreaming>

Some languages provide intelligent arrays which blend the best of both worlds: solid speed, and the ability to quickly append, insert, and delete portions of the array. They're not as fast as IDL's arrays, but they are a whole lot more flexible. And while I'm at it, another array type I'd love to see in IDL is an associative or hash array, preferrably to replace the structure/class, with its rigid rules for adding tags, etc. Very often, you'd like to map a string or other collection of values of any type to another set of values, and you'd like that mapping to change during runtime. At present, you might use a collection of linear arrays, searching through one of them with "WHERE" everytime to index the others. This linear search is very wasteful, and is exactly the sort of thing hashes were designed to solve.

</pieintheskydreaming>

Good luck,

JD
