Subject: Re: How to use pointers instead of arrays
Posted by paul wisehart on Mon, 09 Dec 2002 14:39:35 GMT
View Forum Message <> Reply to Message

```
;create an array
your_array = indgen(10,10)

your_array_pointer = ptr_new(your_array,/no_copy)
;the '/no_copy' causes the ptr to point to the 'ACTUAL' array
;OTHERWISE it would make a copy of it.
;With the '/no_copy' you shouldn't try and access the array w/the
;original variable name 'your_array'

;to access the array
print, (*your_array_pointer)[2,5]
;print the array at position [2,5]
```

Subject: Re: How to use pointers instead of arrays
Posted by David Fanning on Mon, 09 Dec 2002 14:56:44 GMT
View Forum Message <> Reply to Message

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)]
```

When you finish adding things to your array, you trim
it to the correct size:

```
  array = array[0:counter-1]
```

This is both efficient and fast. But do learn about pointers.
They are incredibly useful creatures. They just won't do you
any good here.

Cheers,

David
--
David W. Fanning, Ph.D.
Fanning Software Consulting, Inc.
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

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

Subject: Re: How to use pointers instead of arrays
Posted by MKatz843 on Tue, 10 Dec 2002 18:51:40 GMT
View Forum Message <> Reply to Message

If your arrays are reasonably sized, there's an even easier way to
appean columns or rows to an array. OK, so it's not the best method
for memory management or speed, but for many applications,
it works just fine.

In IDL you can do this:

```
a = [1,2,3]
b = [a, 5]
print, b
    1       2       3       5
or
c = [5, a]
print, c
      5       1       2       3
```

So in 2-D you can do this

```
a = indgen(3,3)
print, a
      0       1       2
      3       4       5
      6       7       8
b = [[a],[9,9,9]]  ; append a row to the end
print, b
      0       1       2
      3       4       5
      6       7       8
      9       9       9
c = [a,transpose([9,9,9])] ; turn a vector into a column and append
print, c
      0       1       2       9
      3       4       5       9
      6       7       8       9
```

The number of brackets is related to how many dimensions you're
working with. You can do all of this in 3, 4, or more dimensions
if you're willing to keep track of the brackets.

Of course, you can replace the original variable with the same command:
a = [a, something].

For memory usage, it's not the best method; but for coding elegance,
it's quite compact. IDL is great for on-the-fly variable definitions.

I hope this helps,
M. Katz