Subject: structures?
Posted by Seb on Thu, 05 Sep 2013 15:26:30 GMT
View Forum Message <> Reply to Message

Hi,

I'm trying to avoid cumbersome loops, and think that using structure arrays or pointers, along with index handling, should help. Say we want to build a table of data for each day in a sequence of julian days. The rows in the table for each day represent a unique time of that day. Now we want to examine a collection of files containing data for a particular date/time, and assign each row to the corresponding row in the table for that day. I envision doing this as follows:

```
n_days=10
a_arr=replicate({idxvar:0.0, table:fltarr(10, 10)}, n_days)
```

where idxvar represents a julian day, and the table contains the time series for that day. We could then loop through each file, examining each row and determining which day and which table row in a_arr the row belongs to. Is there a better way to approach this? My concern is that the tables for each day could be very large if the time step in the time series is very small (say a second), and also there could be a large number of days to build time series for. Is this one of those cases where looping, while horrible, is a more resource-friendly way to deal with this?

Thanks,

--
Seb

Subject: Re: structures?
Posted by Phillip Bitzer on Fri, 06 Sep 2013 14:06:18 GMT
View Forum Message <> Reply to Message

Hi Seb-

It's not clear to me what exactly you're trying/wanting to do. I'm not sure why the "time series for that day" is a 10x10 array. Is it always 10x10? Will each "row" of the table contain the 10 element time series for that time?

Is each date in a different file? Do you know how many files/dates a prior?

In addition, what's the ultimate goal of the analysis? This matters in how you might store the data for easier (quicker) access later, particularly because you mention there may be a lot of data.

Are you asking about looping over the files, i.e., read in each file in a loop? Then yes, you'll have to loop to read each file. But, if you're asking about looping through the data _in_ the file, then there may be better ways.

Subject: Re: structures?
Posted by peterkamatej on Sun, 08 Sep 2013 13:37:16 GMT
View Forum Message <> Reply to Message

Hi Seb,

it seems to me that you might need some dynamic data structure, because there might be a different number of rows needed for each day. Also, a structure array is represented internally as just one "IDL Variable" and it needs to be stored in one solid block of computer memory. This could be quite resource-unfriendly if it's going to be very large (say, hundreds of MB). Especially, if you decide to enlarge the array, a new solid block of memory has to be allocated, the contents copied there and eventually the original memory can be freed.

You could use pointer array instead, but I recommend using the new dynamic data types introduced in IDL 8, HASH() and LIST(). I guess they work internally through pointers, so each part of the large data structure can be at different place in the memory, which is certainly more resource-friendly. However, the way you work with HASH() or LIST() is in many aspect similar to using normal arrays, which is also quite user-friendly (unlike using pointers).

Try looking to the IDL Help at these two data types and see if it suits to you.

Matej

Subject: Re: structures?
Posted by spluque on Wed, 25 Sep 2013 21:57:48 GMT
View Forum Message <> Reply to Message

> Hi Seb,
>
> it seems to me that you might need some dynamic data structure, because there might be a different number of rows needed for each day. Also, a structure array is represented internally as just one "IDL Variable" and it needs to be stored in one solid block of computer memory. This could be quite resource-unfriendly if it's going to be very large (say, hundreds of MB). Especially, if you decide to enlarge the array, a new solid block of memory has to be allocated, the contents copied there and eventually the original memory can be freed.
>
> You could use pointer array instead, but I recommend using the new dynamic data types introduced in IDL 8, HASH() and LIST(). I guess they work internally through pointers, so each part of the large data structure can be at different place in the memory, which is certainly more resource-friendly. However, the way you work with HASH() or LIST() is in many aspect similar to

using normal arrays, which is also quite user-friendly (unlike using pointers).
>
> Try looking to the IDL Help at these two data types and see if it suits to you.

Thank you Matej, hashes did turn out to be a great option for this. Their flexibility is impressive. I am using them to create vectors corresponding to fields in a CSV file. I eventually need to write the data into a new CSV file. I see that the WRITE_CSV procedure can do this, and can take a structure as input. The toStruct method for hashes comes in handy. However, the order of the tags is completely arbitrary. Someone has made available a rather long script (http://code.google.com/p/sdssidl/source/browse/trunk/pro/str uct/reorder_tags.pro?r=72) to re-order tags in a structure, but was wondering whether there is a simpler/better way to do this.

Thanks,
Seb

Subject: Re: structures?
Posted by spluque on Thu, 26 Sep 2013 14:42:55 GMT
View Forum Message <> Reply to Message

On Wednesday, September 25, 2013 4:57:48 PM UTC-5, spl...@gmail.com wrote:

>
>> Hi Seb,
>
>>
>
>> it seems to me that you might need some dynamic data structure, because there might be a different number of rows needed for each day. Also, a structure array is represented internally as just one "IDL Variable" and it needs to be stored in one solid block of computer memory. This could be quite resource-unfriendly if it's going to be very large (say, hundreds of MB). Especially, if you decide to enlarge the array, a new solid block of memory has to be allocated, the contents copied there and eventually the original memory can be freed.
>
>>
>
>> You could use pointer array instead, but I recommend using the new dynamic data types introduced in IDL 8, HASH() and LIST(). I guess they work internally through pointers, so each part of the large data structure can be at different place in the memory, which is certainly more resource-friendly. However, the way you work with HASH() or LIST() is in many aspect similar to using normal arrays, which is also quite user-friendly (unlike using pointers).
>
>>
>
>> Try looking to the IDL Help at these two data types and see if it suits to you.
>
> Thank you Matej, hashes did turn out to be a great option for this. Their flexibility is impressive.
I am using them to create vectors corresponding to fields in a CSV file. I eventually need to write

the data into a new CSV file. I see that the WRITE_CSV procedure can do this, and can take a structure as input. The toStruct method for hashes comes in handy. However, the order of the tags is completely arbitrary. Someone has made available a rather long script (http://code.google.com/p/sdssidl/source/browse/trunk/pro/str uct/reorder_tags.pro?r=72) to re-order tags in a structure, but was wondering whether there is a simpler/better way to do this.
>

Suppose we have three vectors of data of the same length, all of which are in a hash. We want to create CSV files with these vectors, but each file would contain a subset of each vector. This is how I am doing this:

```
keys=['a', 'b', 'c']
n=20L
n_group=5L
ts=hash(keys, list(indgen(n), findgen(n), sindgen(n)))
FOR begi=0L, n - 1, n_group DO BEGIN
  endi=(begi + n_group - 1)
  ts_group=create_struct(keys[0], ts[keys[0], begi:endi])
  FOREACH fld, keys[1:*] DO BEGIN
    ts_group=create_struct(ts_group, keys[where(keys EQ fld)], $
                           ts[fld, begi:endi])
  ENDFOREACH
  write_csv, 'test.csv', ts_group
ENDFOR
```

In this example, we the full hash has three vectors, each with 20 elements, and we want to create 4 files with the same three vectors, but each containing 5 elemeents of the original. We want to keep the original order of the keys. It seems rather contrived to have two loops here. Is there a better way to accomplish this?

Thanks,
Seb

Subject: Re: structures?
Posted by chris_torrence@NOSPAM on Thu, 26 Sep 2013 23:34:16 GMT
View Forum Message <> Reply to Message

On Thursday, September 26, 2013 8:42:55 AM UTC-6, spl...@gmail.com wrote:
> On Wednesday, September 25, 2013 4:57:48 PM UTC-5, spl...@gmail.com wrote:
>

>
>>
>
>>> Hi Seb,
>

>>
>
>>>
>
>>
>
>>> it seems to me that you might need some dynamic data structure, because there might be a different number of rows needed for each day. Also, a structure array is represented internally as just one "IDL Variable" and it needs to be stored in one solid block of computer memory. This could be quite resource-unfriendly if it's going to be very large (say, hundreds of MB). Especially, if you decide to enlarge the array, a new solid block of memory has to be allocated, the contents copied there and eventually the original memory can be freed.
>
>>
>
>>>
>
>>
>
>>> You could use pointer array instead, but I recommend using the new dynamic data types introduced in IDL 8, HASH() and LIST(). I guess they work internally through pointers, so each part of the large data structure can be at different place in the memory, which is certainly more resource-friendly. However, the way you work with HASH() or LIST() is in many aspect similar to using normal arrays, which is also quite user-friendly (unlike using pointers).
>
>>
>
>>>
>
>>
>
>>> Try looking to the IDL Help at these two data types and see if it suits to you.
>
>>
>
>> Thank you Matej, hashes did turn out to be a great option for this. Their flexibility is impressive. I am using them to create vectors corresponding to fields in a CSV file. I eventually need to write the data into a new CSV file. I see that the WRITE_CSV procedure can do this, and can take a structure as input. The toStruct method for hashes comes in handy. However, the order of the tags is completely arbitrary. Someone has made available a rather long script (http://code.google.com/p/sdssidl/source/browse/trunk/pro/str uct/reorder_tags.pro?r=72) to re-order tags in a structure, but was wondering whether there is a simpler/better way to do this.
>
>>
>
>
>
>

>
> Suppose we have three vectors of data of the same length, all of which are in a hash. We want to create CSV files with these vectors, but each file would contain a subset of each vector. This is how I am doing this:
>
>
>
> keys=['a', 'b', 'c']
>
> n=20L
>
> n_group=5L
>
> ts=hash(keys, list(indgen(n), findgen(n), sindgen(n)))
>
> FOR begi=0L, n - 1, n_group DO BEGIN
>
> endi=(begi + n_group - 1)
>
> ts_group=create_struct(keys[0], ts[keys[0], begi:endi])
>
> FOREACH fld, keys[1:*] DO BEGIN
>
> ts_group=create_struct(ts_group, keys[where(keys EQ fld)], $
>
> ts[fld, begi:endi])
>
> ENDFOREACH
>
> write_csv, 'test.csv', ts_group
>
> ENDFOR
>
>
>
> In this example, we the full hash has three vectors, each with 20 elements, and we want to create 4 files with the same three vectors, but each containing 5 elemeents of the original. We want to keep the original order of the keys. It seems rather contrived to have two loops here. Is there a better way to accomplish this?
>
>
>
> Thanks,
>
> Seb

Hi Seb,

If you can wait a couple of months, IDL 8.3 will have a new OrderedHash class, which will preserve the order of the keys. There will also be a new Dictionary class, which forces keys to be case insensitive, but allows you to use "dot" notation to add/retrieve keys, just like a structure.

Cheers,
Chris
ExelisVIS
