
Subject: Re: Releasing memory in IDL

Posted by [Jean H.](#) on Wed, 23 Jan 2008 17:51:58 GMT

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

jaime.e.pineda@gmail.com wrote:

> Hi,

>

> I am trying to run a function (clumpfind) several times with different

> parameters, however, after I call it once not enough memory is

> available. I checked with 'help, /memory' to find variables that I

> could destroy to release memory, without luck (see below).

>

> Is this the expected behavior of IDL? how can I release memory from

> the main function?

>

> Best,

> Jaime

your clumpfind function may be leaking. Try calling heap_gc after the first run

Jean

Subject: Re: Releasing memory in IDL

Posted by [Jaime](#) on Wed, 23 Jan 2008 20:19:42 GMT

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

Thanks Jean,

it helped me... but I was forced to call heap_gc and delete a couple of arrays (a=0) at the end, but still inside, of the clumpfind function. After that I still needed to call heap_gc after clumpfind is called.

It looks still strange to me that I should all this to get some memory back... is there a way to avoid the memory leak?

Best,

Jaime

>

> your clumpfind function may be leaking. Try calling heap_gc after the

> first run

>

> Jean

Subject: Re: Releasing memory in IDL
Posted by [Jean H.](#) on Wed, 23 Jan 2008 20:35:03 GMT
[View Forum Message](#) <> [Reply to Message](#)

Jaime wrote:

> Thanks Jean,
>
> it helped me... but I was forced to call heap_gc and delete a couple
> of arrays (a=0) at the end, but still inside, of the clumpfind
> function. After that I still needed to call heap_gc after clumpfind is
> called.
>
> It looks still strange to me that I should all this to get some memory
> back... is there a way to avoid the memory leak?
>
> Best,
> Jaime

hum...

for regular variables (i.e. not pointers), they are released from memory when you exit the pro or function... so setting "a=0" should change the memory used IN the function, but should have no effect when the program returns to a lower / main level. Indeed, this is correct provided that a) the variable is not part of the argument of the function, or b) the variable is not part of a common block.

To avoid memory leak... well... pointers need to be well defined and cleaned up!

```
a = ptr_new(indgen(10000000000))  
a = 0 ==> you just lost track of the indgen(10000000000) array!!! ...  
but it is still in memory!
```

Jean

Subject: Re: Releasing memory in IDL
Posted by [David Fanning](#) on Wed, 23 Jan 2008 21:02:45 GMT
[View Forum Message](#) <> [Reply to Message](#)

Jaime writes:

> It looks still strange to me that I should all this to get some memory
> back... is there a way to avoid the memory leak?

Better programming always helps! Thanks for the chuckle. :-)

Cheers,

David

--

David Fanning, Ph.D.

Fanning Software Consulting, Inc.

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

Sepore ma de ni thui. ("Perhaps thou speakest truth.")

Subject: Re: Releasing memory in IDL

Posted by [Paul Van Delst\[1\]](#) on Wed, 23 Jan 2008 21:12:21 GMT

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

Jean H wrote:

> Jaime wrote:

>> Thanks Jean,

>>

>> it helped me... but I was forced to call heap_gc and delete a couple

>> of arrays (a=0) at the end, but still inside, of the clumpfind

>> function. After that I still needed to call heap_gc after clumpfind is

>> called.

>>

>> It looks still strange to me that I should all this to get some memory

>> back... is there a way to avoid the memory leak?

>>

>> Best,

>> Jaime

>

> hum...

> for regular variables (i.e. not pointers), they are released from memory

> when you exit the pro or function... so setting "a=0" should change the

> memory used IN the function, but should have no effect when the program

> returns to a lower / main level. Indeed, this is correct provided that

> a) the variable is not part of the argument of the function, or b)the

> variable is not part of a common block.

>

> To avoid memory leak... well... pointers need to be well defined and

> cleaned up!

>

> a = ptr_new(indgen(10000000000))

> a = 0 ==> you just lost track of the indgen(10000000000) array!!! ...

Not completely:

```
IDL> x=ptr_new(lindgen(100))
```

```
IDL> help, x
```

```
X          POINTER = <PtrHeapVar1>
```

```
IDL> help, *x
```

```
<PtrHeapVar1> LONG = Array[100]
IDL> x=0
IDL> help, x
X          LONG = 0
IDL> print, ptr_valid()
<PtrHeapVar1>
IDL> y=ptr_valid(1,/cast)
IDL> help, y
Y          POINTER = <PtrHeapVar1>
IDL> help, *y
<PtrHeapVar1> LONG = Array[100]
```

Though, off the top of my head, I dunno how you would determine the pointer heap variable number programmatically.

cheers,

paulv

Subject: Re: Releasing memory in IDL
Posted by [Paul Van Delst\[1\]](#) on Wed, 23 Jan 2008 21:15:00 GMT
[View Forum Message](#) <> [Reply to Message](#)

Paul van Delst wrote:

```
> Jean H wrote:
>> Jaime wrote:
>>> Thanks Jean,
>>>
>>> it helped me... but I was forced to call heap_gc and delete a couple
>>> of arrays (a=0) at the end, but still inside, of the clumpfind
>>> function. After that I still needed to call heap_gc after clumpfind is
>>> called.
>>>
>>> It looks still strange to me that I should all this to get some memory
>>> back... is there a way to avoid the memory leak?
>>>
>>> Best,
>>> Jaime
>>
>> hum...
>> for regular variables (i.e. not pointers), they are released from
>> memory when you exit the pro or function... so setting "a=0" should
>> change the memory used IN the function, but should have no effect when
>> the program returns to a lower / main level. Indeed, this is correct
>> provided that a) the variable is not part of the argument of the
>> function, or b) the variable is not part of a common block.
>>
```

```

>> To avoid memory leak... well... pointers need to be well defined and
>> cleaned up!
>>
>> a = ptr_new(indgen(10000000000))
>> a = 0 ==> you just lost track of the indgen(10000000000) array!!! ...
>
> Not completely:
>
> IDL> x=ptr_new(lindgen(100))
> IDL> help, x
> X          POINTER = <PtrHeapVar1>
> IDL> help, *x
> <PtrHeapVar1> LONG   = Array[100]
> IDL> x=0
> IDL> help, x
> X          LONG    =      0
> IDL> print, ptr_valid()
> <PtrHeapVar1>
> IDL> y=ptr_valid(1,/cast)
> IDL> help, y
> Y          POINTER = <PtrHeapVar1>
> IDL> help, *y
> <PtrHeapVar1> LONG   = Array[100]
>
> Though, off the top of my head, I dunno how you would determine the
> pointer heap variable number programmatically.

```

Duh! I shoulda kept reading the docs before posting....

```

IDL> print, ptr_valid(count=c)
<PtrHeapVar1>
IDL> print, c
      1

```

Subject: Re: Releasing memory in IDL
 Posted by [Jaime](#) on Wed, 23 Jan 2008 21:28:52 GMT
[View Forum Message](#) <> [Reply to Message](#)

I dug into the code and found that these variables are defined in a COMMON block. Why are they still using memory after the program finished? can they be destroyed altogether (e.g. common_block=0)?

Best,
 Jaime

```

> hum...
> for regular variables (i.e. not pointers), they are released from memory

```

> when you exit the pro or function... so setting "a=0" should change the
> memory used IN the function, but should have no effect when the program
> returns to a lower / main level. Indeed, this is correct provided that
> a) the variable is not part of the argument of the function, or b)the
> variable is not part of a common block.
>
> To avoid memory leak... well... pointers need to be well defined and
> cleaned up!
>
> a = ptr_new(indgen(10000000000))
> a = 0 ==> you just lost track of the indgen(10000000000) array!!! ...
> but it is still in memory!
>
> Jean
