Subject: Re: saving variables between calls to a procedure?
Posted by Reimar Bauer on Wed, 31 Jul 2002 19:08:07 GMT
View Forum Message <> Reply to Message

Shawn wrote:

> Hello,
> I am just curious if there is a better way to save variables
> between a call to a procedure than to define them in a common block
> which is shared between procedures, or passing them back and forth.
> One person suggested to me saving them to a file and re-reading a
> file. But I am hoping for some simple way to declare a particular
> variable as a "saved" variable, which IDL will remember the next time
> the procedure is called. It actually just occured to me that I might
> declare a common block which is actually not shared with any other
> procedure, then I would be able to avoid mistakenly using the variable
> in that procedure in a way that I did not intend to use it. Is it
> possible to do this, declare a common block, but not use it in any
> other procedure, and will IDL remember the variables if the common
> block is not used in any other procedure? Of course it is simple for
> me to test this on my own, so I guess I am really still asking if
> there is a better way.
> Thank you,
> Shawn Young

Dear Shawn,

I don't like common blocks too. Most times I am using parameters
as variable or structure or pointer.
It depends on what's the routine should do.

For example you can define a pointer as heap var or with a value
outside in your main routine.
If you know the reference ptr then you can access from
all routines you want the data behind this pointer.

```
pro define,ptr
 *ptr=[*ptr,10]
end

IDL> ptr=ptr_new(0)
IDL> define,ptr
IDL> PRINT,*ptr
IDL> 0 ,10
```

Reimar

.

Subject: Re: saving variables between calls to a procedure?
Posted by Paul Van Delst[1] on Wed, 31 Jul 2002 21:43:36 GMT
View Forum Message <> Reply to Message

Reimar Bauer wrote:
>
> Shawn wrote:
>
>> Hello,
>> I am just curious if there is a better way to save variables
>> between a call to a procedure than to define them in a common block
>> which is shared between procedures, or passing them back and forth.
>> One person suggested to me saving them to a file and re-reading a
>> file. But I am hoping for some simple way to declare a particular
>> variable as a "saved" variable, which IDL will remember the next time
>> the procedure is called. It actually just occured to me that I might
>> declare a common block which is actually not shared with any other
>> procedure, then I would be able to avoid mistakenly using the variable
>> in that procedure in a way that I did not intend to use it. Is it
>> possible to do this, declare a common block, but not use it in any
>> other procedure, and will IDL remember the variables if the common
>> block is not used in any other procedure? Of course it is simple for
>> me to test this on my own, so I guess I am really still asking if
>> there is a better way.
>> Thank you,
>> Shawn Young
>
> Dear Shawn,
>
> I don't like common blocks too. Most times I am using parameters
> as variable or structure or pointer.
> It depends on what's the routine should do.
>
> For example you can define a pointer as heap var or with a value
> outside in your main routine.
> If you know the reference ptr then you can access from
> all routines you want the data behind this pointer.
>
> pro define,ptr
> *ptr=[*ptr,10]
> end

Hmm. That seems like an extremely dangerous thing to do - couldn't you clobber something

by concatenating like that? If IDL is smart enough to recognise that the next bit of memory may be used by something else it then seems that you would end up with a non-contiguous data structure (in the figurative).

paulv

--
Paul van Delst
CIMSS @ NOAA/NCEP/EMC Beer is good.
Ph: (301)763-8000 x7274 My wife.
Fax:(301)763-8545

Subject: Re: saving variables between calls to a procedure?
Posted by Mark Hadfield on Wed, 31 Jul 2002 22:14:39 GMT
View Forum Message <> Reply to Message

"Reimar Bauer" <R.Bauer(no spam)@fz-juelich.de> wrote in message news:ai9ck5$41hp$1@zam602.zam.kfa-juelich.de...

> Shawn wrote:
>
>> ...It actually just occured to me that I might declare a common
>> block which is actually not shared with any other procedure, then I
>> would be able to avoid mistakenly using the variable in that
>> procedure in a way that I did not intend to use it. Is it possible
>> to do this, declare a common block, but not use it in any other
>> procedure, and will IDL remember the variables if the common block
>> is not used in any other procedure?

Yes. Make sure you give it a name that won't clash with anything else, eg "my_procedure_common"

> I don't like common blocks too. Most times I am using parameters as
> variable or structure or pointer. It depends on what's the routine
> should do.

I don't "like" common blocks either but what on earth is wrong with using one here?

--
Mark Hadfield "Ka puwaha te tai nei, Hoea tatou"
m.hadfield@niwa.co.nz
National Institute for Water and Atmospheric Research (NIWA)

Subject: Re: saving variables between calls to a procedure?

Posted by David Fanning on Wed, 31 Jul 2002 22:26:48 GMT
View Forum Message <> Reply to Message

Paul van Delst (paul.vandelst@noaa.gov) writes:

>> pro define,ptr
>> *ptr=[*ptr,10]
>> end
>
> Hmm. That seems like an extremely dangerous thing to do - couldn't you clobber something
> by concatenating like that? If IDL is smart enough to recognise that the next bit of
> memory may be used by something else it then seems that you would end up with a
> non-contiguous data structure (in the figurative).

This doesn't seem dangerous to me (perhaps because
I use the construct all the time). It seems like
one of those wonderful things IDL occasionally does
that makes you think to yourself "Now, by God, that's
how software *ought* to work!"

In any case, it works, over and over and over. And it
never occurred to me that non-contiguous data storage
could be involved, even remotely. Not only that, I
don't even have pointer memory leaking like a sieve,
which--I think--is the totally miraculous part. :-)

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: saving variables between calls to a procedure?
Posted by David Fanning on Wed, 31 Jul 2002 22:28:03 GMT
View Forum Message <> Reply to Message

Mark Hadfield (m.hadfield@niwa.co.nz) writes:

> I don't "like" common blocks either but what on earth is wrong with
> using one here?

I'm afraid I have to agree. If ever there was a perfect
time to use a common block, the original poster described

it.

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: saving variables between calls to a procedure?
Posted by Mark Hadfield on Wed, 31 Jul 2002 23:12:46 GMT
View Forum Message <> Reply to Message

"David Fanning" <david@dfanning.com> wrote in message
news:MPG.17b21214ff7d9b35989944@news.frii.com...

> Paul van Delst (paul.vandelst@noaa.gov) writes:
>
>>> pro define,ptr
>>> *ptr=[*ptr,10]
>>> end
>>
>> Hmm. That seems like an extremely dangerous thing to do - couldn't
>> you clobber something by concatenating like that? If IDL is smart
>> enough to recognise that the next bit of memory may be used by
>> something else it then seems that you would end up with a
>> non-contiguous data structure (in the figurative).
>
> This doesn't seem dangerous to me (perhaps because I use the
> construct all the time). It seems like one of those wonderful things
> IDL occasionally does that makes you think to yourself "Now, by God,
> that's how software *ought* to work!"
>
> In any case, it works, over and over and over. And it never occurred
> to me that non-contiguous data storage could be involved, even
> remotely.

Extending an array a with the a = [a,b] syntax doesn't create a non-contiguous data structure. What is does is create a new array a, insert the elements from the existing a and b into it, then delete the old a. This is fine for small arrays but it slows down on large arrays because of all the memory allocation & deallocation. It is *very* bad practice to create a large array by growing it one element at a time.

What do I mean by "large" in this context. I don't know, a few thousand I guess. Here is an exercise for the reader: time the following code for various values of n:

```
a = [0]
for i=1,n-1 do a = [a,0]
```

--
Mark Hadfield "Ka puwaha te tai nei, Hoea tatou"
m.hadfield@niwa.co.nz
National Institute for Water and Atmospheric Research (NIWA)

Subject: Re: saving variables between calls to a procedure?
Posted by JD Smith on Thu, 01 Aug 2002 01:38:51 GMT
View Forum Message <> Reply to Message

On Wed, 31 Jul 2002 16:12:46 -0700, Mark Hadfield wrote:

> "David Fanning" <david@dfanning.com> wrote in message
> news:MPG.17b21214ff7d9b35989944@news.frii.com...
>
>> Paul van Delst (paul.vandelst@noaa.gov) writes:
>>
>>>> pro define,ptr
>>>> *ptr=[*ptr,10]
>>>> end
>>>
>>> Hmm. That seems like an extremely dangerous thing to do - couldn't you
>>> clobber something by concatenating like that? If IDL is smart enough
>>> to recognise that the next bit of memory may be used by something else
>>> it then seems that you would end up with a non-contiguous data
>>> structure (in the figurative).
>>
>> This doesn't seem dangerous to me (perhaps because I use the construct
>> all the time). It seems like one of those wonderful things IDL
>> occasionally does that makes you think to yourself "Now, by God, that's
>> how software *ought* to work!"
>>
>> In any case, it works, over and over and over. And it never occurred to
>> me that non-contiguous data storage could be involved, even remotely.
>
> Extending an array a with the a = [a,b] syntax doesn't create a
> non-contiguous data structure. What is does is create a new array a,
> insert the elements from the existing a and b into it, then delete the
> old a. This is fine for small arrays but it slows down on large arrays
> because of all the memory allocation & deallocation. It is *very* bad

> practice to create a large array by growing it one element at a time.
>
> What do I mean by "large" in this context. I don't know, a few thousand
> I guess. Here is an exercise for the reader: time the following code for
> various values of n:
>
> a = [0]
> for i=1,n-1 do a = [a,0]

I just realized a factor of two increase in speed in a case of very repetitive use of the a=[a,b] mechanism with vectors of length 10 or less! So yes, a=[a,b] concatenation is very slow when compared to setting aside the memory to begin with, and just incrementing an index. Unfortunately, you don't always know how long your array will eventually be (luckily, in my problem I had a reasonable upper bound on the number). Decent languages provide dynamic arrays which pre-allocate memory in increasing blocks, and thus avoid the overhead of malloc() on every round, not to mention maintaining flexible array endpoints to keep push and shift operations O(1). It's just absolutely ridiculous to copy the array everytime you'd like to add something to it.

In "fast" languages like C without advanced memory handling ala dynamic arrays, this type of problem is usually solved by a linked list, for which addition, insertion, and deletion are all O(1) operations. Unfortunately, the high overhead of IDL's pointers (which are nothing like machine-level C pointers) limit the utility of this method too (although I guess I should test that statement).

Anyway, the lesson is, beware of a=[a,b], which is convenient, but costly.

JD

Subject: Re: saving variables between calls to a procedure?
Posted by David Fanning on Thu, 01 Aug 2002 04:13:56 GMT
View Forum Message <> Reply to Message

JD Smith (jdsmith@as.arizona.edu) writes:

> I just realized a factor of two increase in speed in a case of very
> repetitive use of the a=[a,b] mechanism with vectors of length 10 or less!

Alright, I concede that this is probably not fast, although the vectors I use this technique with always have fewer than 100 elements, and I've always found it to be fast enough. (I just shake my head at the reports of people who have tried it with vectors of length 10 bazillion or so, and have been disappointed.)

But I stand by my statement that it's neat. :-)

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: saving variables between calls to a procedure?
Posted by Reimar Bauer on Thu, 01 Aug 2002 08:32:11 GMT
View Forum Message <> Reply to Message

David Fanning wrote:

> JD Smith (jdsmith@as.arizona.edu) writes:
>
>> I just realized a factor of two increase in speed in a case of very
>> repetitive use of the a=[a,b] mechanism with vectors of length 10 or
>> less!
>
> Alright, I concede that this is probably not fast,
> although the vectors I use this technique with always have
> fewer than 100 elements, and I've always found it to
> be fast enough. (I just shake my head at the reports
> of people who have tried it with vectors of length
> 10 bazillion or so, and have been disappointed.)
>
> But I stand by my statement that it's neat. :-)
>
I am too.

And it was only an example ...

regards

Reimar

> Cheers,
>

> David
>

Subject: Re: saving variables between calls to a procedure?
Posted by Paul Van Delst[1] on Thu, 01 Aug 2002 14:45:52 GMT
View Forum Message <> Reply to Message

Mark Hadfield wrote:
>
> "David Fanning" <david@dfanning.com> wrote in message
> news:MPG.17b21214ff7d9b35989944@news.frii.com...
>
>> Paul van Delst (paul.vandelst@noaa.gov) writes:
>>
>>>> pro define,ptr
>>>> *ptr=[*ptr,10]
>>>> end
>>>
>>> Hmm. That seems like an extremely dangerous thing to do - couldn't
>>> you clobber something by concatenating like that? If IDL is smart
>>> enough to recognise that the next bit of memory may be used by
>>> something else it then seems that you would end up with a
>>> non-contiguous data structure (in the figurative).
>>
>> This doesn't seem dangerous to me (perhaps because I use the
>> construct all the time). It seems like one of those wonderful things
>> IDL occasionally does that makes you think to yourself "Now, by God,
>> that's how software *ought* to work!"
>>
>> In any case, it works, over and over and over. And it never occurred
>> to me that non-contiguous data storage could be involved, even
>> remotely.
>
> Extending an array a with the a = [a,b] syntax doesn't create a
> non-contiguous data structure.

I'm not worried about extending arrays like a=[a,b], but pointers ala *ptr=[*ptr,10]. If ptr is pointing to some data structure (scalar or array), it seems feasible to me that the next position in memory could be occupied by something else. If one then does a *ptr=[*ptr,10], what happens to what was in that memory? Is it "protected" somehow so that the value "10" is then placed in the next "free" spot in memory giving a non-contiguous array [not efficient, but safe], or are the contents clobbered and replaced by the value "10" [very very bad], or are the contents moved to some other memory location so that the original position can be used to store the value "10" [relatively efficient(?) and safe].

> What is does is create a new array a,
> insert the elements from the existing a and b into it, then delete the

> old a.

So does this mean that

*ptr = [*ptr,10]

is equiavalent to:

a = *ptr
ptr_free, ptr
a=[a,10]
ptr=ptr_new(temporary(a))

?

(Which was my original question, just poorly phrased)

paulv

--
Paul van Delst
CIMSS @ NOAA/NCEP/EMC Beer is good.
Ph: (301)763-8000 x7274 My wife.
Fax:(301)763-8545

Subject: Re: saving variables between calls to a procedure?
Posted by David Fanning on Thu, 01 Aug 2002 15:03:44 GMT
View Forum Message <> Reply to Message

Paul van Delst (paul.vandelst@noaa.gov) writes:

> So does this mean that
>
> *ptr = [*ptr,10]
>
> is equiavalent to:
>
> a = *ptr
> ptr_free, ptr
> a=[a,10]
> ptr=ptr_new(temporary(a))
>
> ?

I think the answer to this question is "yes".

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: saving variables between calls to a procedure?
Posted by JD Smith on Thu, 01 Aug 2002 17:59:14 GMT
View Forum Message <> Reply to Message

On Thu, 01 Aug 2002 07:45:52 -0700, Paul van Delst wrote:

> Mark Hadfield wrote:
>>
>> "David Fanning" <david@dfanning.com> wrote in message
>> news:MPG.17b21214ff7d9b35989944@news.frii.com...
>>
>>> Paul van Delst (paul.vandelst@noaa.gov) writes:
>>>
>>>> > pro define,ptr
>>>> > *ptr=[*ptr,10]
>>>> > end
>>>>
>>>> Hmm. That seems like an extremely dangerous thing to do - couldn't
>>>> you clobber something by concatenating like that? If IDL is smart
>>>> enough to recognise that the next bit of memory may be used by
>>>> something else it then seems that you would end up with a
>>>> non-contiguous data structure (in the figurative).
>>>
>>> This doesn't seem dangerous to me (perhaps because I use the
>>> construct all the time). It seems like one of those wonderful things
>>> IDL occasionally does that makes you think to yourself "Now, by God,
>>> that's how software *ought* to work!"
>>>
>>> In any case, it works, over and over and over. And it never occurred
>>> to me that non-contiguous data storage could be involved, even
>>> remotely.
>>
>> Extending an array a with the a = [a,b] syntax doesn't create a
>> non-contiguous data structure.
>
> I'm not worried about extending arrays like a=[a,b], but pointers ala
> *ptr=[*ptr,10]. If ptr is pointing to some data structure (scalar or

> array), it seems feasible to me that the next position in memory could
> be occupied by something else. If one then does a *ptr=[*ptr,10], what
> happens to what was in that memory? Is it "protected" somehow so that
> the value "10" is then placed in the next "free" spot in memory giving a
> non-contiguous array [not efficient, but safe], or are the contents
> clobbered and replaced by the value "10" [very very bad], or are the
> contents moved to some other memory location so that the original
> position can be used to store the value "10" [relatively efficient(?)
> and safe].

The best way to think of pointers in IDL is to forget what you know from other languages like C, and regard them as special unnamed, but otherwise perfectly normal IDL variables which live in a global heap available everywhere, and thus aren't created/destroyed when you enter or exit routines.

```
IDL> a=ptr_new('test')
IDL> delvar,a
IDL> help,/heap
Heap Variables:
   # Pointer: 1
   # Object : 0

<PtrHeapVar1>   STRING    = 'test'
IDL> resurrected=ptr_valid(1,/cast)
IDL> print,*resurrected
test
```

Here we see we can recall the pointer heap variable off the heap. And just as for normal IDL variables, you can arbitrarily reassign them, extend them, change the data type of, etc.:

```
IDL> b=ptr_new('x')
IDL> *b=10.1
IDL> *b=[*b,10000]
IDL> print,*b
      10.1000      10000.0
```

This is exactly analogous to:

```
IDL> b='x'
IDL> b=10.1
IDL> b=[b,10000]
IDL> print,b
      10.1000      10000.0
```

The only difference between "b" and "*b" is the former has a name, and a scope (in the $MAIN$ level here), whereas the latter is global, and only

accessible through a pointer. In both cases we were able to change from a string to a scalar floating value to a two index array.

Of course, this ease of type/size/layout changing hides lots of behind-the-scenes memory-juggling, some of which may be rather inefficient, so it pays to have a little more familiarity with the methods these operations use internally.

JD

Subject: Re: saving variables between calls to a procedure?
Posted by JD Smith on Thu, 01 Aug 2002 18:04:42 GMT
View Forum Message <> Reply to Message

On Wed, 31 Jul 2002 21:13:56 -0700, David Fanning wrote:

> JD Smith (jdsmith@as.arizona.edu) writes:
>
>> I just realized a factor of two increase in speed in a case of very
>> repetitive use of the a=[a,b] mechanism with vectors of length 10 or
>> less!
>
> Alright, I concede that this is probably not fast, although the vectors
> I use this technique with always have fewer than 100 elements, and I've
> always found it to be fast enough. (I just shake my head at the reports
> of people who have tried it with vectors of length 10 bazillion or so,
> and have been disappointed.)
>
> But I stand by my statement that it's neat. :-)

It is neat, very neat. I use it all the time, and rarely have to optimize around it (here it was occuring hundreds of thousands of times...). Growable arrays make life so much easier, obviating chunkier implementations , like linked-lists, for almost all cases. Imagine how much neater it would be, however, if not only could you grow your arrays (on either end), but such operations would occur in O(1) time, independent of the array size. This is an array feature already present in a variety of other languages.

JD

Subject: Re: saving variables between calls to a procedure?
Posted by Paul Van Delst[1] on Thu, 01 Aug 2002 18:38:45 GMT
View Forum Message <> Reply to Message

JD Smith wrote:
>
> On Thu, 01 Aug 2002 07:45:52 -0700, Paul van Delst wrote:
>
>>
>> I'm not worried about extending arrays like a=[a,b], but pointers ala
>> *ptr=[*ptr,10]. If ptr is pointing to some data structure (scalar or
>> array), it seems feasible to me that the next position in memory could
>> be occupied by something else. If one then does a *ptr=[*ptr,10], what
>> happens to what was in that memory? Is it "protected" somehow so that
>> the value "10" is then placed in the next "free" spot in memory giving a
>> non-contiguous array [not efficient, but safe], or are the contents
>> clobbered and replaced by the value "10" [very very bad], or are the
>> contents moved to some other memory location so that the original
>> position can be used to store the value "10" [relatively efficient(?)
>> and safe].
>
> The best way to think of pointers in IDL is to forget what you know from
> other languages like C, and regard them as special unnamed, but otherwise
> perfectly normal IDL variables which live in a global heap available
> everywhere, and thus aren't created/destroyed when you enter or exit
> routines.

Ahh. Fair enough.

> Here we see we can recall the pointer heap variable off the heap. And
> just as for normal IDL variables, you can arbitrarily reassign them,
> extend them, change the data type of, etc.:

I did not know you could do that with pointers in IDL.

> Of course, this ease of type/size/layout changing hides lots of
> behind-the-scenes memory-juggling, some of which may be rather
> inefficient, so it pays to have a little more familiarity with the methods
> these operations use internally.

Yeah.... apart from inspecting data files, I use IDL mostly to prototype code for Fortran 90/95 so I tend to avoid (or, as in this case, not even know about <ehem>) the neato power-user type of tricks.

paulv

--
Paul van Delst
CIMSS @ NOAA/NCEP/EMC Beer is good.
Ph: (301)763-8000 x7274 My wife.
Fax:(301)763-8545

Subject: Re: saving variables between calls to a procedure?
Posted by David Fanning on Thu, 01 Aug 2002 19:13:23 GMT
View Forum Message <> Reply to Message

Paul van Delst (paul.vandelst@noaa.gov) writes:

> Yeah.... apart from inspecting data files, I use IDL mostly to prototype code for Fortran
> 90/95 so I tend to avoid (or, as in this case, not even know about <ehem>) the neato
> power-user type of tricks.

Paul, I know you have the community service portion of
the IDL-EPA application filled out pretty completely,
but--man--this is not the kind of thing that is going
to endear you with the blow-hards down at the pub! :-(

Couldn't you say that you use IDL for *something*
useful! I'm pushing for you, man, but give me some
help 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: saving variables between calls to a procedure?
Posted by Paul Van Delst[1] on Fri, 02 Aug 2002 14:33:01 GMT
View Forum Message <> Reply to Message

David Fanning wrote:
>
> Paul van Delst (paul.vandelst@noaa.gov) writes:
>
>> Yeah.... apart from inspecting data files, I use IDL mostly to prototype code for Fortran
>> 90/95 so I tend to avoid (or, as in this case, not even know about <ehem>) the neato
>> power-user type of tricks.
>
> Paul, I know you have the community service portion of
> the IDL-EPA application filled out pretty completely,
> but--man--this is not the kind of thing that is going
> to endear you with the blow-hards down at the pub! :-(
>

> Couldn't you say that you use IDL for *something*
> useful! I'm pushing for you, man, but give me some
> help here.

Sorry man, I don't know what to tell you. Maybe if I shout the blowhards down at the pub a round they'll think kindly of me?

paulv

--
Paul van Delst
CIMSS @ NOAA/NCEP/EMC Beer is good.
Ph: (301)763-8000 x7274 My wife.
Fax:(301)763-8545
