Subject: heap and stack
Posted by John Persing on Wed, 16 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

I will present a hypothesis, then a line of evidence, then I will wait for somebody else on the group to post a more sensible explanation.

Hypothesis:
There are three type of variables: stack variables, temporary variables, and heap variables. (Plus a fourth type of abstraction of a "variable" which stores information about user-defined variable types like structures and objects, which we need not worry about here.) All are of the same format: a structure of metadata that includes a reference to the memory location of the first element of the data. The three forms differ only in their metadata.

Example:

```
IDL> a = [5, 6, 7]
IDL> help, a
A               INT       = Array[3]
IDL> b = PTR_NEW(TEMPORARY(a))
IDL> help, b
B               POINTER   = <PtrHeapVar1>
IDL> help, a
A               UNDEFINED = <Undefined>
IDL> help, *b
<PtrHeapVar1>   INT       = Array[3]
IDL> print, b
<PtrHeapVar1>
IDL> print, *b
      5       6       7
```

A stack variable counts the number of its own occurrences. A temporary variable probably has the most primitive metadata structure. Pointers are organized into a master, global list.

In the first assignment "a = [5, 6, 7]", first "[5, 6, 7]" is converted to a primitive, temporary variables allocating a piece of memory to store the array. Then a stack variable "a" is created with "1" instance and the memory location is transcribed to "a". Then the temporary metastructure is destroyed.

In the second assignment "b = PTR_NEW(TEMPORARY(a))", first "TEMPORARY(a)" creates a temporary metastructure and transfers the metadata from "a" to this, including the reference to the data. Then the metadata in "a" is set to appear as an undefined variable, and the number of instance is set to "0", but something in TEMPORARY circumvents the ordinary behavior of stack variables to wipe out the data in memory when its occurrences are set to

"0". Then the pointer metadata is created, the metadata from the temporary variable is transfer along with the memory location, and the temporary metastructure is destroyed. A metstructure for "a" must always remain. The metastructure for "b" is a stack variable, but it makes reference to no memory location instead some other piece of metadata in the "b" metastructure makes reference to "<PtrHeapVar1>".

An alternate explanation is that TEMPORARY does not set number of occurrences of "a" to "0", just set the memory reference in "a" to a null reference. Then, the only time that stack variables do their automatic clean is at the end of functions and procedures and the only times that the number of occurrences of a stack variable changes is at definition, at the start of a procedure, at the end of a procedure.

Thanks for any help. The reason I want this depth of detail is that I am going to pretend to be an expert in it at a group presentation. :)

--
}3 John Persing }3
http://www.frii.com/~persing persing@frii.com
Half of all Americans earn less than the median income!!!!!!

Subject: Re: heap and stack
Posted by davidf on Thu, 17 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

John Persing (persing@frii.com) presents as good a theory
of IDL variables as any I've seen, and then writes:

> The reason I want this depth of detail is that I am
> going to pretend to be an expert in it at a group presentation. :)

Uh, sorry. I've got to, ah...get a cavity filled that day.
I won't be able to make it. :-(

Cheers,

David

P.S. Scheesh! Let's just say there are some HUGE
advantages to working on the beach. :-)

--
David Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/

Subject: Re: heap and stack
Posted by Vapuser on Thu, 17 Jun 1999 07:00:00 GMT
View Forum Message <> Reply to Message

"John Persing" <persing@frii.com> writes:

I think you're getting too complicated. If you have the 'idl advanced development guide' of something like that, it has a discussion of what constitues an IDL variable. The 'Cliff Notes' on it is that the variable is a large descriptor with flags describing what the variable is, a Union allowing for the storage of scalars and the appropriately constructed structures allowing for the storage of dynamically allocated data structures. All variable types are accounted for, heap, pointer, object, array, structure and any kind of scalar. So there is no need a 'stack', 'temporary' or 'heap' variable. They're all just IDL_VARIABLES. Discrimination is made by virtue of the flags set within this descriptor.

See the file $IDL_DIR/idl/external/export.h for the skinny on this. Look at the IDL_TYP_* macros, they define the type, and the IDL_ALLTYPES union, which stores that data of the variable. You'll see there is a macro defined for pointers and object type variables and a flag for 'Heap' variable within the structure. The discrimination of whether a variable is temporary, constant, a file, a structure or dynamically allocated is made by querying the state of the IDL_VARIABLE 'flags' field.

Still, having said all that, your explanation 'sounds' more 'expert' ;> How are they to know?

William

> I will present a hypothesis, then a line of evidence, then I will wait for
> somebody else on the group to post a more sensible explanation.
>
> Hypothesis:
> There are three type of variables: stack variables, temporary variables,
> and heap variables. (Plus a fourth type of abstraction of a "variable"
> which stores information about user-defined variable types like structures
> and objects, which we need not worry about here.) All are of the same
> format: a structure of metadata that includes a reference to the memory
> location of the first element of the data. The three forms differ only in
> their metadata.
>

(snip)

> --
> }3 John Persing }3
> http://www.frii.com/~persing persing@frii.com
> Half of all Americans earn less than the median income!!!!!!

Yes, but half earn more. I hope I'm one of them.

--
William Daffer: 818-354-0161: vapuser@catspaw.jpl.nasa.gov
