

Re: Array comparison

Source: <http://coding.derkeiler.com/Archive/Delphi/alt.comp.lang.borland-delphi/2004-10/0104.html>

From: Rob Kennedy (*me3_at_privacy.net*)

Date: 10/08/04

Date: Fri, 08 Oct 2004 15:00:13 -0500

Nicolai Hansen wrote:

>> *Nicolai Hansen wrote:*

>>> *Deep or shallow comparison?*

>>

>> *Shallow. (Read on for details.)*

>

> *Deep and recursive. You want to see if they are equal, don't you?*

Yes. But I also want to be able to see if two pointer types are equal. Does pointer variable A point to the same thing as pointer variable B? Is the address stored in A the same as the address stored in B? With deep compares, I can't do that since the compiler will be dereferencing the pointers and comparing their referents instead of comparing the pointers.

And once you start doing deep comparison for types that support cross references (such as object), you run the risk of never-ending recursive comparisons.

>> *You can't have cross references in static arrays or records, so there's*

>> *nothing to fear on that front.*

>

> *No, but if we were to allow general comparison between any type, we*

> *also need to be able to compare objects.*

I'm not asking for any changes to existing operators. The "=" operator already works on object references by comparing their pointer values.

> *My records and classes should be compared field by field. But all*
> *pointers should have the objects on which they point compared. Deep*
> *comparison, and recursive. How else would you want ^integer to be*
> *compared? By address?*

Of course. If I wanted to compare their referents, then I would have dereferenced them before applying the "=" operator. TThing and PThing are separate types, and they should be treated that way.

> *if integer(a)=integer(b)? whats hard about that?*

For one thing, it's a non-intuitive type cast. (Casting two like-typed variables to a different type just to compare them? No thanks.) For another, it assumes that pointers and integers are the same size.

>> *The compiler already performs an element-wise copy for arrays; why not
>> an element-wise compare? If you're going to argue against comparing
>> arrays, then I think you also need to argue against assigning arrays.*
>
> *Does it really copy element wise?*

Yep.

> *Doesn't it just use a memcpy function for this?*

Nope. See for yourself. It's in System._CopyArray. (While you're there, take a look at _CopyRecord, too.) Simple elements get copied with System.Move; non-simple elements get copied using the usual assignment semantics (notably, reference counting).

> *Think about the confusion about comparing open arrays of chars;*

I haven't considered open arrays yet. They're not static arrays, at least not as far as their typeinfo is concerned. They're not exactly dynamic arrays, either, since they have no reference count and their lengths are stored separately from the rest of the data. I'd have to look more closely at what happens when an open array is passed from one function to another before I could say how it should be compared.

> *null-terminated strings returned from API calls etc etc etc.*
> *'abcdef#0'a' _is_ equal to 'abcdef#0'b' when returned from the*
> *API..*

If you treat them as generic pointers, then they are not equal since their addresses differ. If you treat them as arrays of characters, then they are not equal since their eighth elements differ. To treat them as null-terminated strings, you need to pass them to a special comparison function, like StrComp, which will tell you that the two are equal. And that's consistent with what I'm asking for. If you want something other than the compiler's default comparison, you can write your own, but there should at least *be* a default comparison available.

> *Also, think about a variant record.*

>

> *TTest=record*

> *case bit16: Boolean of*

> *True: (w: word);*

> *False: (b: array[0..1] of byte);*

> *end;*

>

> *You can easily assign two instances of this record to eachother. But*

> *how would you compare two of those? This would be "comparing apples to*

> oranges" which simply is not possible.

They're of the same type, so compare them the same way they were assigned. _CopyRecord will use Move to assign that record, so it can be compared using CompareMem. The variant portion of a record will always be comparable with CompareMem because it's not allowed to contain types like string or dynarray. Variant records don't present any difficulty.

> *I'm not saying that its not possible to compare two arrays; not even
> dynamic arrays. I'm just stating that it would (imo) be wrong to
> implement comparison of some user defined types, but not all. And also
> that it might cause some unwanted trouble.*

I think I understand. You're saying that if arrays and records are comparable, then objects should also be comparable -- more than just by reference -- right?

But if objects are going to be compared deeply, then they should be assigned deeply, and that will break everything. Everything.

--
Rob