

Re: Array comparison

Source: <http://coding.derkeiler.com/Archive/Delphi/alt.comp.lang.borland-delphi/2004-10/0090.html>

From: Rob Kennedy (*me3_at_privacy.net*)

Date: 10/07/04

Date: Thu, 07 Oct 2004 11:46:03 -0500

Nicolai Hansen wrote:

>>Apply the "=" operator to each of the array's elements.

>

> And if the element is an object, you compare addresses, but if its an

> integer you compare values?

Yes. The same as what already happens when you compare two non-array variables. The following code should always be compilable, no matter the value of T:

```
var
  X, Y: T;
begin
  X := Y;
  Y := X;
  if X = Y then ;
end;
```

OK, "no matter the value" is a little strong. I haven't given much thought to whether comparisons should be allowed for Delphi's native "file" types. They're not assignable, so I'm less inclined to say they should be comparable. (Aside: Does *any* operator work on file types?)

> My reasoning is as follows:

>

> *You cannot compare arrays.

You're begging the question. That's the point you're supposed to be arguing, so it's no fair to start your reasoning with that premise.

> "Array of Byte" and "Array of Byte" can

> not be compared in Pascal; you need to declare them as type.

I have no problem with the compiler considering separately declared types to be incompatible with each other. What I have a problem with is that two variables of identical types may be incompatible. If two variables are assignable, then they should be comparable.

And I don't want the focus to shift from static arrays to dynamic arrays. The compiler already knows how to compare dynamic arrays, so within this message, when I talk about "arrays," I'm talking exclusively about static arrays.

- > **You could then compare your types, but how on earth could you make*
- > *something logical that can compare two types if they can compare only*
- > *array types, and not record types?*

That's a good point. I thought I'd mentioned records already, but maybe I didn't. I want records to be comparable, too. It's hard to argue for one without the other.

- > **Next step would be to compare any instance type with another instance*
- > *of the same type.*
- > *–All strings can be compared to eachother (compares the contents of*
- > *the string, but not the reference counts)*
- > *–All TObjects can be compared (shallow comparison, compares only the*
- > *address of the objects)*

OK. I agree with that.

- > **What to do with all other types?*

You mean like Integer, Double, Variant, and IUnknown? The compiler already knows how to compare those.

- > *Deep or shallow comparison?*

Shallow. (Read on for details.)

- > *Deep and*
- > *recursive comparison (fear the cross references!)? Who is to tell me*
- > *how my records should be compared?*

You can't have cross references in static arrays or records, so there's nothing to fear on that front.

Why shouldn't your records be compared field-to-field? What other meaningful comparison is possible? If all the fields of one record are not identical to the corresponding fields of another record, then I would not consider the records to be equal in the first place. If you want to compare for "almost equal," then it begins to look like what happens when comparing two floating-point values — you need to come up with your own routine to decide **how** equal the values need to be.

- >> *The only valid point in that is records. The compiler already knows how*
- >> *to compare everything else, although some of the comparisons (e.g.,*
- >> *objects, pointers) aren't always the most useful. But that the compiler*
- >> *doesn't know how to provide an intelligible comparison of two objects*
- >> *does not explain why it can't compare two like-typed arrays.*

>
> *You cannot allow comparisons of certain user-defined types, but not of*
> *others.*

Good thing that's not my platform. You **can** compare two objects, but all you get is a comparison on their references. Arrays aren't reference types. They're value types, so compare their values. (The "value" of an array, in this case, would be the composition of all its elements' values.)

> *The fun part here is that IIRC two identical types can be compared*
> *according to the Pascal ISO.*

Are you saying that the ISO calls for the same thing I'm calling for?

>> *Untyped pointers are very easy to compare. People do it all the time.*
>>
>> *var*
>> *a, b: Pointer;*
>>
>> *if a = b then ...*
>
> *Yes, it is easy to compare the value of the pointers. IE compare them*
> *as if they were integers... But this might not be what was wanted.*

That's OK with me. The compiler should perform as shallow a comparison as possible. It's easy to write your own routine to go deeper when necessary. If the default comparison is deep, though, it's harder to write your own routine to do a shallow or mid-depth compare. The problem is that arrays and records have **no** comparison.

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.

>> *Whatever the reasons for disallowing "=" on static arrays and records,*
>> *technical difficulty is not one of them -- not nowadays, anyway. I'm*
>> *more inclined to say the reason is lack of customer demand for the feature.*
>
> *I never said it was a technical difficulty in comparing arrays;*

Well, you did make claims about what the Pascal compiler is capable of doing.

> *I stated that it was plain easy to write your own compare methods. My*
> *point was that I see reason in it NOT being allowed, due to confusion*
> *about which types can and can not be compared.*

I see more confusion in the status quo. I'm looking for fewer exceptions to the rule that you can use the "=" operator to compare two variables. Having fewer exceptions leaves less room for confusion.

alt.comp.lang.borland-delphi: Re: Array comparison

It is indeed easy to write your own comparison operations. It's also tedious — something a compiler is perfectly suited to doing.

--

Rob