

Re: Poll: SizeOf(variable) vs SizeOf(type), should SizeOf(variable) be banned?

Re: Poll: SizeOf(variable) vs SizeOf(type), should SizeOf(variable) be banned?

Source: <http://coding.derkeiler.com/Archive/Delphi/alt.comp.lang.borland-delphi/2008-01/msg00409.html>

- *From:* Rob Kennedy <me3@xxxxxxxxxxx>
 - *Date:* Tue, 08 Jan 2008 10:05:19 -0600
-

Skybuck Flying wrote:

Rob Kennedy wrote:

That doesn't fix anything. Delphi already behaves that way.

I know that's what I ment.

Your claim is that by forbidding SizeOf on variables, and forcing SizeOf on types only, the compiler will catch errors. That's simply not true.

The opposite is also not true, as shown above.

Right. So you've taken a situation where mistakes are possible, introduced an arbitrary rule, and we're left with a situation where all the same mistakes are possible. Thanks.

In other words:

Using SizeOf on variables does not prevent programming mistakes.

I never said it did.

The common case:

SizeOf(on pointer variables)

How will you solve the problem for pointer variables ?

You can't use SizeOf on pointer variable because that will return the size of the pointer, that is not what you want, you want the size of

Re: Poll: SizeOf(variable) vs SizeOf(type), should SizeOf(variable) be banned?

the array.

No, when I call SizeOf on a pointer variable, it's because I want to know how big the variable is. If I wanted to know the size of the thing the pointer variable is declared to point to, I would have called SizeOf on something else, or perhaps referred to some other variable that holds the size of the thing the pointer really points to (as opposed to its declared value).

I don't use pointers to point to arrays. For that, I use dynamic arrays, which case I use Length in conjunction with SizeOf to determine the amount of memory occupied by the array data (on the rare occasions when I actually need that information).

What if the pointer is untyped in that case you have no choice but to use SizeOf on a type.

Why in the world would I be using an untyped pointer? And what could I possibly expect to get when calling SizeOf on the dereferenced pointer?

Alternatively you would need to make the pointer a typed pointer by adding a Ptype so no matter how you look at it, you still need to create an additional type, might as well have done that in the first place.

Yes. Introducing types is very good. It helps you tell the compiler what you're doing.

Conclusion:

Using SizeOf on types is the only way to get it working in all cases.

Just because you had to introduce a typed pointer doesn't mean you have to call SizeOf on that type. You can call it on the dereferenced pointer just as easily. Perhaps even `_more_` easily as I explained in my first reply to this topic, since calling SizeOf on a dereferenced pointer variable means you don't need to go find what the declared type of the variable is. The compiler already knows, so let the compiler do the work.

I want to see an example where calling SizeOf on a variable leads to a mistake that would have been fixed by calling SizeOf on a type instead.

Mostly besides the point.

The point is:

There are now two ways to do the same thing, one method probably less

Re: Poll: SizeOf(variable) vs SizeOf(type), should SizeOf(variable) be banned?

Re: Poll: SizeOf(variable) vs SizeOf(type), should SizeOf(variable) be banned?

universal, why add this potential confusion.

Probably? So you're not even sure this new rule would always work.

By the way, there is one situation in Delphi where calling SizeOf on a variable is the only thing you can do — it's impossible to call SizeOf on that variable's type.

And what would that be ?

I'll give you a hint. It has to do with a certain kind of array.

I have already shown opposite cases:

Calling SizeOf on untyped pointer variables to determine array length
<- flawed.

Calling Abs to get the square root of a number is also flawed. So what?

Calling SizeOf on dynamic array variables to determine array length <-
flawed.

Calling StrToInt to get the capital of Russia is also flawed. So what?

However why would somebody dynamically allocate a static
array
instead of using a dynamic array in the first place. Doesn't
make
too much sense.

When did I say the static array was being allocated dynamically? If
it were, the same code wouldn't have compiled when the array type was
changed to dynamic.

What reason could you possibly have then for calling SizeOf ?

I've called SizeOf to get the size of an *element* of an array. But rarely to get the size of an array itself.

If I had a static array of characters and was passing it to an API function, I would pass the length of the array, not the size. Unicode API functions expect their buffer lengths measured in characters, not bytes.

Re: Poll: SizeOf(variable) vs SizeOf(type), should SizeOf(variable) be banned?

Re: Poll: SizeOf(variable) vs SizeOf(type), should SizeOf(variable) be banned?

Static arrays on the stack are a bad idea anyway, think
"buffer
overrun".

Who said anything about the stack?

Well that leaves few possibilities:

static arrays in record.
static arrays in objects/classes.

OK. There are also static arrays in the EXE's data section. That is, global and local typed constants.

—
Rob
.