

Re: ///Wannabee Rants

Source: <http://coding.derkeiler.com/Archive/Assembler/alt.lang.asm/2007-01/msg00324.html>

- *From:* "///o///annabee <Free\" " <Wannabee@xxxxxxxxxxxxx>
 - *Date:* Tue, 09 Jan 2007 14:40:23 +0100
-

På Tue, 09 Jan 2007 06:57:46 +0100, skrev rhyde@xxxxxxxxxxxxx <rhyde@xxxxxxxxxxxxx>:

///o///annabee <Free" wrote:

This can also be done with RosAsm macro system, checking for sizes of labels. and registers as well, I believe.

Really? Why don't you show us, then?

Not that I could ever imagine any need for it.

Sevag just gave you an example of why it's useful. Read and learn.

I am reading and reading, and it seems none of you are really able to explain to me why this typechecking is so important, and neither how it works. You seem to repeat that I dont know what it is, and I am asking you, just because I dont know. You should at least know exactly what it is, and be able to show/explain all its uses, so that it will make clear sense to everyone why this choise is so important.

sofar, you have mentioned distinguishing types at compiletime, and keeping programmer from making typing errors. Is this all that it can do?

Still it will not protect from a perverted register,

No, type checking won't catch every possible semantic error out there. That's why it's called "type checking" rather than "all possible error checking."

Very well. But it must clearly have more uses that those listet to be warranted in an assembler.

Re: ///Wannabee Rants

so what is its
advantage that warrant all this code mess ?

Because it **can** catch many errors.

And the result is **far** more
readable to someone who is not intimately familiar with the code, but
needs to modify it for their own purposes.

I dont think so at all. Actually this is some of the worst I have seen in readability departement. Plus the example Sevag showed was just a couple of lines. In something real this would be much longer, and an endless pain to read. As I said, you seem to have reinvented spaghetti, because you have overcomplicated your syntax.

Your code is very much more
verbose, killing readability.

Wrong, dude. Conciseness, especially at the level that RosAsm
programmers tend to operate, is what kills readability.

huh? How the bleeding macaroni does "conciseness" kill readability?
Thats like saying a mathematical expression should not be simplified as it would kill readability. Thats virtually an insane suggestion.

This is imho like having a cloud around you
head when coding.

You seem to forget that code is written **once** but read **many** times.

You contradict yourself. You disputed this yourself just above when you said :
"but
needs to modify it for their own purposes."

Yep, having to explicitly specify things makes the code a bit more
difficult to **write**.

Re: ///Wannabee Rants

Re: ///Wannabee Rants

And this will at least quadruple for large projects.

But reading, understanding, and maintaining the code, especially by someone other than the original author, is much easier when the code doesn't look like an explosion in an alphabet soup factory (the best way to describe most RosAsm programs I've seen).

But that's exactly what your code does look like. Reinvention of spaghetti caused by overcomplication of a simple code that is much clearer in its original form.

Very brutal and very plain facts.

I've seen your code. Granted, René's is worse, and maybe a/b's as well.

Fully wrong. Mind you I have modified some of René's code, and it never took much time to do that. And he rewrites large segments of his own code, over and over, making it clearer and clearer. We evolve our writing styles, over time you know.

But you'll have a hard time convincing people around here that your code is readable.

I am not trying to. It's much too obvious that they will see this for themselves.

Should imo be
forbidden in programming.

Look in the mirror.

Actually, your code would probably become more readable if I mirrored it.

This is nothing but reinventing spagetticode.

???

You don't even know what the term (spaghetti code) means, do you?

Okey, then call it macaroni. Who cares, it is obscured way beyond any kind of reasonable meaning.

Re: ///Wannabee Rants

Re: ///Wannabee Rants

Look at my plain asm code. It is tons better from all points of view.

People around here have. You've posted lots of code over the years. Mostly it gets ignored. And when it doesn't, the comments haven't been pretty good.

For the code I posted latly, all the partisipants, having some programming background, understood exactly.

```
Proc SaveObject:  
Argument @BaseObject  
mov edi D@BaseObject  
BaseObject.vCall vSaveObject  
EndP
```

First of all, on a trivial code sequence like this, it is impossible to get a feeling for what your writing style is like. Nonetheless, the code above is still unreadable. Though that is more due to RosAsm bizarre syntax than your coding style.

If this is unreadable to you, then it a stretch to imagine you can be much of a programmer.

To do what. You havent told us what it can do. In addition to distinguishing types. Distinguishing types is easily done with a callable, the way I described in my post to betov. Thats what objects are for.

Yeah, like Rene you would do things at run-time that really ought to be done at compile-time.

Thats what OOP is invented for. To do runtime-linking. (Late time binding).

And you have the nerve in other posts to talk about how assembly language has all the advantages over assembly while writing such poor code.

This sentance doesnt make any sense.

- > For the above, you would first need a "Proc" macro that can determine
- > what types of data to accept.

Re: ///Wannabee Rants

Re: ///Wannabee Rants

This is by default done. It takes an object. Only if you pass it a perverted pointer will it fail. And typechecking cant guard againts this. You said so yourself.

You really need to learn what the term "type checking" means. You embarass yourself with your ignorant comments.

Thats why I am asking you. You seem to have reinvented typechecking in HLA, so you should at least be able to explain it. Cant figure it could be that difficult for the man that implemented such a feature, and calling it so important, to explain to us why it is such a nice and important feature. You're a teacher. Go on. Teach us!

```
> In Rosasm case, It's practically already
> in the syntax, all you would need is for the macro system to recognize
> "Argument" items as types that you have declared elsewhere:
> It would be roughly:
>
> [@BaseObject 4] ; define "BaseObject" as type with 4 bytes
>
> [myClassPtr @BaseObject] ; reserve 4 bytes for myClassPtr as
> "BaseObject"
```

This isnt RosAsm valid syntax. Actually crash RosAsm. And such a simple syntax...

So why offer it as an example? Why not give us real stuff that actually works rather than fantasy wish lists?

You dont pay much attention. All the >> > confusing you allready ? :)
This is Sevag you are replying to

```
> Proc SaveObject:
> Argument @BaseObject ; declare parameter 1 as type "BaseObject"
> mov edi D@BaseObject
> BaseObject.vCall vSaveObject
> EndP
>
> invoke SaveObject, myClassPtr
>
> Now if you passed the wrong pointer type, the macros processor would
> know and give an error.
```

Re: ///Wannabee Rants

And how would it do that?

Why don't you explain how this works rather than just making stuff up that sounds plausible? Give us a real example; one that won't "crash" RosAsm.

You dont pay much attention. All the >> > confusing you allready ? :)

This is Sevag you are replying to, again! This is he who wrote what you are replying to.

So this is a compiletime feature trying to prevent a sleepy programmer from making a bug?

I can't really tell. I don't see how this would do what you're claiming. Why not explain it to us?

You dont pay much attention. All the >> > confusing you allready ? :)

This is Sevag you are replying to. Third time.

- > But this is a simple use of type checking. A
- > more significant use, as demonstrated earlier is when a macro can do
- > various things according to the type of data passed to it.

What we have objects for.

No, that's not the purpose of objects. Once again, you're trying to use run-time structures to take care of things that should be handled statically, at compile time.

I can't figure out why parsing thousands of userdefined types, many times over in `_every_` function, maybe several times in each, in a program maybe having several thousand, if not 100000s of functions, could be a better strategy. While even for millions of vtable calls, linking at runtime, theres a 1 cycle penalty for dynamic linking. Per call. My computer now is able to perform 2,0 BILLION cycles per second. Because of the taxing so infrequent, this 1 cycle penalty will literally vannish in the noise. Possible even nullified by the CPU pipeline. At the great benefit of much more comprehensive code, no need for slow compilations, and much better overall program structure. Seem much harder to justify this complex beast that your code is becomming, when implemented in a real program.

- > For
- > example, my "buffer" class can accept input data from many different
- > pointers: signed/unsigned 8/16/32/64/128 (converted to ascii), char,
- > all standard types (as data bytes), strings, streams of data with
- > address offsets, another buffer class, etc.

Re: ///Wannabee Rants

Some of this is also possible with RosAsm macros. but what is a typical use?

Take a look at the HLA stdout.put macro sometime.

Good grief. No way!

I cant see why. Is this extendable to userdefined types?

Yes.

Then the distaster is factuall. There could be thousands and thousand of those, all over the place in a huge project. And the compiler has to distinguish every type, in every piece of the code, many times per function, sometimes many times per line. Now I understand why your compiler took several seconds to compile a 5 liner when tested over a year ago. This will scale extremely badly into a real world project.

Scanning all those types could take lots of time in a large project.

At compile time, when it should be done, not at run time. Some day, I hope you figure out what the difference is.

This is the diffrence between very bad design and very bloated code (with typechecking) and the clean and easy code I post, plus the excellent streamlined execution times of my code.

Thats why OOA was invented, to do this directly via callable.

No, that's not why OOA was invented. If you think so, it demonstrates that you don't have a clue about object-oriented programming. Not surprising, as you don't even know what type checking is about.

The only true benefit of OOP is for run time linking. Otherwise its just an extention to the use of records. Plus also the very easy to maintain code structure that naturally develop if not abused.

fair enough. This has been debated to death. Thats exactly why I try another angle. I am trying to understand the _advantage_ of type checking,

The first, you should study programming language design and **learn** what type checking is all about.

Re: ///Wannabee Rants

Re: ///Wannabee Rants

No I am asking you. You claim its such an advantage that it must be had. So thats why you should explain to us, why this is so. If you cant, you admit that you dont know, and then your statement that this is such a *must have* feature, is based on a misconception, and thus cannot be taken seriously.

One would have thought that with the 10 years of Delphi experience you've claimed to have, you'd know all about type checking (as well as object-oriented programming). That doesn't appear to be the case.

Thats exactly what I have been telling you (5 times) allready. Why the frickin bong, would I ask if I allready knew for certain? And why you keep stating this 5 times, while I have confirmed it also 5 times. Are you a retard or something?

in order to understand why it is important enough for you to require it by an assembler.

When you learn about the benefits of type checking, you'll probably want it in assembly language as well :-)

Why the **** do you think I ask you????

What I have understood now, by master PDFs and your posts, is that it is for catching mistyping bugs.

That is one thing it's good for. It's also good, as Sevag as pointed out, as a tool for directing code generation during macro expansion to generalize those macros.

Okey. Good. Then I was correct. This is not needed with OOA and polymorphism. This confirms I was right, and that I do not need this in asm. Thanks, finally we got some answers. To bad you could not cut this shorter, by just stating that at once.

And maybe also a primitive form of OOA that does not scale.

It has nothing to do with OOA.

No. True. It is what OOA takes care of in a much more logical way.

Re: ///Wannabee Rants

Re: ///Wannabee Rants

And I suspect you don't understand what the term "to scale" means, either.

Scale well means that overabuse will not cause any serious effects. Or that at least when the feature is used excessively in a large project it doesn't slow things down too much. But your feature does do that, and OOA does not. So that's why I say your typechecking feature, does not scale. This can be seen by anyone, if they turn on or off typechecking in Delphi. A program with 100000 lines, takes `_minutes_` to compile with typechecking on, and takes about 2 secs with typechecking off. This is for Delphi 7.

Cheers,
Randy Hyde