

Re: The War On HLA

Source: <http://coding.derkeiler.com/Archive/Assembler/alt.lang.asm/2003-10/0718.html>

From: Randall Hyde (*randyhyde_at_earthlink.net*)

Date: 10/31/03

Date: Fri, 31 Oct 2003 06:09:24 GMT

> *I guess. For an assembler to be able to build highlevel like syntax macros
> would be neat, but I mean, a call would do about the same there too. A
> powerful macrosystem may be neat and clean, but its more a matter of
> preference than it adds real power to a language. What is their real
> diffrence in what they can do ?*

Take a look at the HLA compile-time language. The stuff you can do with it simply cannot be done by procedures and functions.

> *Delphi supports most stackframes, and also procedures without any
> stackframe, write them in basm, and they are what ? Macros, no less. The
> call thats whats the difference ? IMO macros doesnt allways make for better
> reading. They are unneeded spesial cases.*

Macros can be abused just like any other language feature. So in that respect you're absolutely correct -- they don't **always** make for better reading. I find it amusing, however, that HLL programmers (e.g., Delphi, C/C++, etc.) will go on and on about how great abstract **data** types are, but then complain that user-created control structures (i.e., abstract control structures) are a terrible idea. Seems like a very weak argument to me.

>>
>> *In sections of code where you need raw speed, you need to be careful
>> about using black boxes (e.g., macros). However, if speed is that
> important,
>> I would ask why you're using Delphi in the first place.
>
> If you have to ask, you shouldnt. It shows that you havent spent enough time
> with it.*

I've spent a lot of time with it. Raw speed and small projects are not Delphi's forte. Delphi is great for rapid development of GUI apps.

>
> *Oh common Randall. Allows you to extend the language ? This is *very* weak
> argument. Are you saying that functions and procedure does NOT allow you to*

alt.lang.asm: Re: The War On HLA

- > *extend the language ? What are you saying then ? Cause that statement is just hillerious.*

Procedures and functions allow you to extend the language one way. Macros let you extend it another. For example, try and create your own context-free control structure using procedures or functions — it can't be done (this is why, for example, you can't create true iterators in Delphi). A decent macro system allows you to do this. For example, the HLA language doesn't have a SWITCH (CASE) statement. Yet I can write a macro with HLA's compile-time language that supports a very powerful switch statement. Try and write a CASE statement in Delphi using procedures (and, of course, without using Pascal's CASE statement). True, Pascal doesn't need such a statement, but there are many others that it does need. And you can implement them with a procedure or function call. With a decent macro subsystem, however, you could.

- > *If the optimizer is *far* from great, explain how its not so great. Do you just have to debate everything, even when you're on thin ice ? the optimizer works pretty well, when given a little help. Its the same with all compilers, they need a litle inside information help. If you make call to properties of objects inside a loop, it suck, yes, but there are ways to write godd HLL and bad HLL. Good HLL is optimized pretty decent. Good pascal programmers write ASM comparable speed when put to the test. I have seen this in the basm group about 5 years ago. Or was it here in fact ?*

This is a well-known fact. If you don't realize that Delphi's code generator is inferior to many other compilers, I suggest you take a look at some of the many compiler benchmarks floating around. In general, Borland is not well-known for their compiler optimization strategies (this includes their C++ compilers as well as Delphi). MSVC++ generates far better code. GCC generates better code. And *those* compilers are not generally included among the best optimizing compilers available. Delphi does an *okay* job, but it's not great by any stretch of the imagination. Even a mediocre assembly language programmer can beat Delphi's code generation on a regular basis.

- >
- > *No Randy. You are too much of a proffessor. Any ability in a language, any feature is left to the programmer to invent his own uses for. You can declare a constant data, and a variable, point the variable to the const data and do what you want. Its just data anyway. An OOP programmer doesnt have MUCH need for static variables in any case, he converts dynamic memory to "static" memory when he needs it. Its just the same memory.*
- > *A pascal programmer has many types of data he can declear, the memory are just memory, he can access it any way he wants to, you dont (yet) get a warning from M\$ for accessing you data a little diffrently. If the memory section is writeable, you can write to it.*

In a few years, when you get a little more programming experience under your belt, you'll need to come back and read this statement for a good laugh.

"Why yes! I'll declare a const object so I can *write* data to it; makes perfect sense. People will be able to read and understand my programs with ease!"

There are a lot of OOP programmers out there who are going to be surprised that they have no need for static variables...

> *What I meant is that when you write huge programs meant to accomplish a great deal, you dont hardcode everything in the program. in fact the less the better. You can do it of course, but its much easier to NOT hardcode things. It makes for better possibilities for user interaction. And modifications. "No stupid user, you cannot change this bitmap, its hard coded in the application source.". "What do you want to rotate that spere 180 degress in realtime ? You want it to be half the size, wait for the next release, it will feature a sphere rotated exactly as you want it, and shruken too."*
> *Oh Randy no need to save my word documents anymore, just hardcode them in Office. (Hell is it that the reason they come out so often and are using so much diskspace) ?*

And you're claiming I'm making weak arguments?

With a resource file, you get to make four copies of the data!

For some things, that approach is well worth it. For an initial value of some abstract data type that is not visible to the outside world, this is a waste. And it still doesn't explain why it makes sense to read and *write* const objects in a language. In Delphi, const has almost no meaning because of this "feature" in the language. It is non-intuitive and dangerous (the compiler *should* give a warning when you write a statement that stores data into a const object).

>
> *Oh, this is getting boring. Can you leave that stupid constants now? Its just data Randy. They dont issue a certificate yet on how to access it. And we hardly use them anyway.*

It's not just data. It's *read-only* data.

But Delphi has the surprising, and non-intuitive, action of allowing you to *write* to *read-only* data. That's a defect in the language design. Pure and simple. The solution is simple: add a "static" keyword to the language and support static objects. Obviously they felt this feature was necessary or they wouldn't allow you to create structured constants in the first place (and, *horrors* write to those const objects).

>> *And it would be nice to have the loader preinitialize that data so you don't have to waste code and time doing it once the program starts up. As the stuff is not "constant" data, it doesn't seem write to declare such variables in Pascal's "const" section.*
>
> *A program need some constants. I prefer to have as litle as possible.*

That's your programming style. Good, bad, or indifferent, you've got to realize that other people have other programming styles. Most people

tend to use lots of constants (named constants). Most people expect that the compiler will prevent them from creating statements that write to those constants. It's not just data, it's *read-only* data. The fact that your programming skill might not have yet matured to the point where you see the need for such facilities in a language doesn't mean that the need doesn't exist.

- > *My*
- > *table object for example, can contains so rediculously many items and*
- > *columns that I prefere to make the MaxItems a variable. Its a 64bit integer*
- > *variable and has an implisit MAX. Maxint64 is predefined in delphi and no*
- > *need to declare it again. Its utterly dynamic, containing one item or an*
- > *insane not even imageable amount, on an int64 count of pages for the insane*
- > *harddisk listing. I truely never have to tough that code again. I try to*
- > *write low level code to never do it again.*
- > *But some things are unavoidable to decleare as contants. Predeclaring all my*
- > *5000 mp3 songs inside a program is on the otherhand just weird. And somehow*
- > *I think that you know this pretty well. I stopped writing hello world*
- > *applications many years ago.*

Some things are constants, some are not. I'm not debating that. What I'm claiming is that some languages (and Delphi is not among them) provide a reasonable way to create data tables (constant or variables) that are preinitialized when you load the program into memory. The constant data is *constant* and cannot be changed, the variable data is variable and can change under program control. Delphi confuses these two concepts.

- > > *Microsoft's "thunks" are a small category of the general concept of*
- > > *a thunk. A thunk is a piece of code and its execution environment*
- > > *rolled into a variable. One way to think of a thunk is as a variable that*
- > > *can hold a statement (or sequence of statements) for later execution.*
- > > *This allows you to treat statements as "first-class" objects in a*
- > > *programming language (i.e., you can assign them to variables,*
- > > *return them as function results, do various operations on them, etc.).*
- > > *Note that a procedure does not qualify because it doesn't carry*
- > > *the execution environment around from where the thunk was*
- > > *created.*
- >
- > *I am not sure what you are talking about.*

You need to read the chapter on thunks in AoA.

- > *Objects/classes/encapsulation/overloading ?*
- > *Cout and stuff like that (C). If thats the case there is nothing new about*
- > *it, and it certainly has nothing to offer in terms of functionality, but for*
- > *a programmer yes, it hides information. Which can be bad. The programmer is*
- > *not perfect and he will die or quit. The code will reveal shortcommings and*
- > *or bugs. The really good thing about OOP is not implementation hideing, but*
- > *scope and modularity. Many of the other features, like the so once popular*
- > *polymorfism and inheritance is mostly the emperors clothing. The real*
- > *strength of OOP is that it allows a single programmer to create huge*

alt.lang.asm: Re: The War On HLA

> *applications given time and effort, and to slowly build a house with
> patience and persistens. But if he doesnt do it ALL himself, he will run
> into problems one day. And it will be something created by some other
> person. That might be dead, or left, or just not available. So he might just
> save himself the effort and do it asm from the start, cause soon he gonna
> hide things even from himself. Thats why I like Betovs approach. And I
> belive he knows all this from experience, and that is why he is so strict
> about it. I belive his on the right track.*

It's clear you're a fan of Betov's approach.

But at the same time, you're talking about writing code that others can read and maintain. That is **not** compatible with Rene's approach to software engineering. Rene still develops software the way it was done in the 1960's. In the 30+ years that have passed since then, we've learned quite a bit about writing maintainable and readable code. Monolithic applications written using "cut & paste" techniques have been out of favor since the 1970's. If you're really interested in writing code that others can read, understand, and maintain, the **last** thing you want to do is follow Rene's lead.

> *Your are not up to date on Delphi. Delphi is the mother of OOP. Not the true
> mother, but the language that made it clear that it had something to offer.
> Borland C++ was created in parallell. They are now virtually identical in
> terms of OOP and interfaces, COM support.(Which I by the way hate like the
> plague)*

Delphi is a hybrid OOP language, just like C++. Hardly a "mother" version of OOP. Ask any **real** OOP programming, and they'll tell you that Simula-67 or Smalltalk is the mother OOP language. The truth is, C++ is a **big** kludge, and Delphi is heavily based on C++'s object facilities. Sad, but true. I'm not saying Delphi's OOP facilities aren't useful, I'm just saying that your statement makes you sould a little foolish and it's not something you want to repeat around long-time object-oriented programmers.

> >
> *Okey Randall. I will read it. I will anyway. And as I recall I have allready
> read it, but it was a long long time ago. Theres been some applications in
> the making between then and now. And my brain is just not big enough to hold
> on to all that information. You must have a really big head.*

No. But I do write stuff down and go back and read it when I need to recall what I knew at one time. Then again, there **are** a lot of people around here who believe I **do** have a fat head :-)

>
> *Yes. For any man. "The list could go on an on."? Seems like its endless. Do
> you really mean that it is endless ?
> Not even figuratively speaking could you say that it is endless.*

Sure. Because I can keep on creating new language features which Delphi obviously doesn't support. Practically speaking, however, Delphi is still a long ways away from a "perfect" language.

>

> *It has ALL the features that a language needs, but not all that you need.*

> *You need a "truck to carry a stawberry". Just like betov says.*

It has rotten string handling facilities (please, don't debate this --- just take a look at SNOBOL4 or Icon if you want to see a language with *decent* string handling facilities).

It doesn't have very good set processing capabilities; again, take a look at SETL if you want to see good set processing capabilities.

It doesn't even have good *array/matrix* handling facilities (look at APL and even certain versions of BASIC).

Lists? Doesn't compare at all to LISP.

> > *The truth is, no matter how long*

> > *the list gets, I can still make this claim.*

>

> *So if the list has to items, it could still go on and on ?*

Yep.

There are hundreds of languages out there. With more being invented every day. Most of these languages has features that Delphi lacks. The list can go on and on and on and ...

> > *Delphi is far from perfect*

> > *(from a language design point of view); I just chose to cut the list*

> > *short for space reasons.*

>

> *Sure. you have a real hard time making yourself verbose :-)*

Believe me, I could easily fill a several hundred page book with language features that Delphi does not possess.

> *And you shouldnt since its all there. What the freaking jonson is*

> *non-imperative programming facilities ?*

Languages like Prolog, LISP, Flex, and Bison are good examples of non-imperative languages.

> *You really are the the PDF jedi. The Obi wan PDF. Lol.*

See, you really are taking after Rene.

> > *I'm sorry I don't have the time to write you a 500 page book*

> > *explaining all the nice language features Delphi *could* have but*

> > *doesn't. However, such books already exist (not to mention lots*

> > *of journal papers), so there really is no need to repeat that information*

> > *here. Go grab one of those books and read about it for yourself*

alt.lang.asm: Re: The War On HLA

> > *if you're interested. Making statements like "No, it couldn't" is a very*
> > *weak statement, even from someone like you :-)*
>
> :-) *Cool. Can you help me out with a title. It seems better to ask you.*
> *You'll know.*

Almost any college text book with "programming language design" in the title would be a good start. When you get done reading the chapters on iterators, thunks, and parameters in AoA, I'll be more than happy to provide you with a reading list if you're still interested.

> >
> > *The fact that *you* don't use it doesn't mean it isn't needed.*
>
> *No. The fact that Basm can perform the actions for me does.*

The fact that you **think** BASM can perform those actions simply means that you're not doing sufficiently advanced programming to realize what the limitations are.

>
> *So you just embed the EBX register in the "thePoint" is that the point ?*
> *My point was that basm code can entail a register like EBX, pointing to a*
> *structure with*
> *the Structure syntax and . Member.*
> *e,g mov edi, [eax].TPDFStudents.Name*
> *now edi points to a string inside the TpdfStudents record/(STRUCT = C).*
> *Furthermore it goes a long way on commenting the code at the same time.*
> *Your code above doesnt do the same, but close. You seem to access a (local)*
> *variable*
> *instead of a pointer in a register. Basm do that too of course. And I never*
> *claimed HLA*
> *couldnt do it. I asked if it could.*
> *When I translated the first line to quasi HLA i found it so easy just to*
> *copy it to the next line, and forgot to*
> *correct the code to "relect" the code I landed. But you could only read the*
> *one with the parantesis ?*

Again, you're missing the whole point.

If all you need to do are two MOV statements, BASM is fine.

For simple assembly sequences surrounded by a lot of Pascal code, BASM is perfect. **I** use it all the time when I need just a few machine instructions to do something and it's not worth writing a whole function or procedure in assembly language to achieve something. Once you need to write some serious assembly code, however, BASM falls on its face. Ask yourself, would you want to write a complex assembly application completely in BASM? Well, maybe **you** would, but a serious assembly programmer would laugh at that prospect. It's just not an appropriate tool. It **is** one of the best in-line assemblers around, but it's not a good assembler when you compare it against the likes of HLA, MASM, TASM, etc.

alt.lang.asm: Re: The War On HLA

>
> *Smile. True, not much of code there. I guess you dump edge scanning code on
> your students ?*
> *I have written 100K lines of half crap/half good code in Delphi. But it
> works. I have only did very few pieces in asm. I have only one section of
> code that I really am proud of. It was a great feeling. Like if you made
> your own sneakers , and wore proud to wear them and show them of. **
> *I have made crap asm code to. Asm is not hard in itself (surprisingly simple
> infact). As I have said before, any language is but a tool. There are
> secrets, you only learn by doing it, and that sets aside the weak asm
> programmer (moi) from the really strong one. Like Betov or perhaps Beth or
> Annie. But syntactically/symbolic asm is the easiest of all languages to
> learn.*
> *But to make something useful of it is of course a hard work, and needs lots
> of thinking to get it really good. So in that sence its the hardest laguage,
> but to repeat, its syntax is simple, and many intructions you can just skip
> to learn, since the more simple instructions can take you a long way.*

Yes, assembly syntax is easy to learn.
It's the semantics that are the bitch.

That's why HLA is proving popular with beginning assembly programmers, BTW.
They get to rely on HLL semantics during the early part of their assembly education.

cheers,

Randy Hyde

P.S. You'll forgive me for not continuing this thread beyond this post. But as I am obviously boring you, plus the fact that

you're headed down the dangerous path of Betov/Engineering, there really isn't much more to say about this topic. I encourage

you to pick up a decent book on software engineering as well as programming language design and check out this stuff for yourself sometime.