

Re: the necessity of Lisp's Objects?

Source: <http://coding.derkeiler.com/Archive/Lisp/comp.lang.lisp/2008-01/msg01646.html>

- *From:* Tim X <timx@xxxxxxxxxxxxxxxxxxx>
 - *Date:* Sun, 27 Jan 2008 16:02:39 +1100
-

Xah Lee <xah@xxxxxxxxxxx> writes:

Jon Harrop wrote:

«The Mathematica language specification does not specify the complexities of most core operations so it is not possible to work out the complexity of a given function. So it doesn't matter how well you know the language, you still cannot ascertain how long your function will take to run.»

Interesting.

well... but what could be some examples? For example, for complex math functions such as Integrate or Solve, you can't know the time behavior in advance. And for simpler things like Sort, basically one can assume it has time behavior of the current knowledge.

I don't think that's true. Different languages implement routines in different ways and have different orders of complexity. You cannot assume a language has used the most efficient algorithm – you may hope they have, but you cannot guarantee it. There have been many times I've found the sort routine of a language to be poorly implemented or based on an 'easy' implementation rather than one that was efficient.

thinking about this... i don't remember Perl, Python, PHP, emacs lisp, Javascript's manuals ever talk about a function's time complexity.
(maybe once or twice)

(For Integrate, Solve, or other complex functions, one should know the time complexity of course since there's the source code, but my guess is that when the function is sufficiently complex, it's impractical to actually know. I mean, say a function that are thousands of lines or calls other libs ... when is the last time you actually do analysis to know a big subroutine's complexity? When you use a complex regex in perl, do you actually do mathematical analysis just so you know what's your function's time complexity?

Re: the necessity of Lisp's Objects?

Agreed you may not know the complexity or efficiency of something prior to using it. However, if you find the performance is not sufficient, you will need to work out or at least estimate that complexity in order to know how to optimise it. Without this critical bit of information, your optimisation is going to be hit and miss.

Basically, if you coded it, you deal with the complexity issue at the time of creating an algorithm, then after that if the program runs well of any reasonable tests, it's good to go. That's how software are practically developed.)

And if it isn't, you examine the algorithm and work out how to reduce its time/space complexity to make it more efficient.

While I don't disagree with the premise that programming languages should, as far as possible, remove the need for the programmer to know about many of the low level implementation aspects. However, I don't agree with your argument that discussions and investigation/analysis of things like types, call stacks, memory heap etc are irrelevant and just 'tech geek jargon and rubbish'. It is exploration, experimentation and theoretical discussion/debate in these areas that extend our understanding and which may, in time, allow us to develop programming languages that further reduce/remove the need for the programmer to know about the lower level aspects of a language they are using.

There is an analogy here with other occupations. For example, years ago when I used to do a lot of woodwork/carpentry, I needed to know about the various properties and characteristics of different types of wood. It wasn't sufficient for me to just know how to use wood working tools well. Likewise, to be a good programmer, it's not just sufficient to know how to use your language tool in an abstract sense. You need to know about the platform you are working on to understand how the language has been implemented on that platform. The type of CPU, its registers, command set, cache, amount of RAM etc can all impact on the performance of the program. Without this information, you can not know the best optimisation strategies.

having said all of that, I also should point out that these days the vast majority of programs people are writing rarely push the boundaries of modern platforms. When I first started programming, CPU speeds and available RAM were slow/small enough that it wasn't uncommon to have to do things like unroll loops and code them in assembler. C was very popular at this time because you could more easily get down and dirty and do very efficient code using pointer arithmetic and customized memory management that was tweaked to better suit the specific requirements or usage pattern of the application you were writing.

Re: the necessity of Lisp's Objects?

Re: the necessity of Lisp's Objects?

These days, this isn't as necessary. This is partly due to improvements in hardware, but also due to improvements in compiler design and optimization – improvements that have been largely achieved due to research, debate and experience with all that tech geek stuff you seem to think was only introduced to try and make the whole field somehow elusive or exclusive and mystify things for the non-initiated. While, in some areas, there has been a tendency to define new terms and concepts when existing ones would have been sufficient, there is also plenty of areas where no existing terminology was sufficient to clearly describe/explain a concept.

Of course, the easy way to prove your point would be to actually develop a language that avoids all the unnecessary terms and concepts which seem to frustrate you. If your right, the language will take off and revolutionise the field.

Tim

—

tcross (at) rapttech dot com dot au

.