What's in a name?

In the once upon a time days of the First Age of Magic, the prudent sorcerer regarded his own true name as his most valued possession but also the greatest threat to his continued good health, for—the stories go—once an enemy, even a weak unskilled enemy, learned the sorcerer's true name, then routine and widely known spells could destroy or enslave even the most powerful. As times passed, and we graduated to the Age of Reason and thence to the first and second industrial revolutions, such notions were discredited. Now it seems that the Wheel has turned full circle (even if there never really was a First Age) and we are back to worrying about true names again.
–True Names, V. Vinge

One of those moments

I was talking to a networking-researcher friend [1] one day, and I suddenly remarked on what a large percentage of our conversation had to do with *names*. He looked at me and said: “Well… yeah. Naming things is one of the fundamental things computer scientists do.”

The light bulb that went off in my head was so blinding, I dropped out of the conversation for about 20 or 30 seconds of dead silence while I tried to deal with his statement.

But, really, I've been dealing with it ever since. I had this conversation around 1993, and since then, it seems that the concept of names has come up over and over in my life working with information architectures.

Distal access and symbol systems

The best explanation of names I ever got—what they are and why they matter—didn't come from a programming-languages person, or a networking person. It came from a scientist who worked in Artificial Intelligence: my thesis adviser, Allen Newell [2].

Newell and Simon's Turing award cited them for their work on “symbol systems.” Newell once told me that this was just names, and then he explained his understanding of names: “They provide distal access.” That is, a *name* is a local piece of data that *stands for* some other piece of data, which is presumably large and remote. You can now use that small, convenient, local datum instead of the large, remote thing for which it stands. The key act you perform with a name (that is, a symbol) is ship it to that remote location, and get back the chunk of data it named. Newell said the career-making, fundamental “aha” experience of his entire life was realising that computers were not, as was typically held in the 1960's, “number crunchers.” They were *symbol processors*—something much more general. They processed names.

If you accept Newell's definition, you suddenly start seeing names everywhere, at every scale:

- Moving a **register id** from the instruction-fetch unit on one side of a CPU to the register bank on the other;
- Shipping an **address** from the CPU to the memory system;
- Referencing a **variable** in a tight loop that was bound on entry to the containing procedure;
- Sending a **host name** to a DNS server down the hall;
- Sending a **URL** from my web browser to a server on the other side of the planet;
- Using a ten-kilobyte **BitTorrent file** to download a multi-gigabyte movie off the net.

These are all just names and their associated dereferencing operations. Any time you see a system that has a notion of “cookies” or “handles:” those are just different names for names.

As far as Newell was concerned, this is the purpose served by names, or symbols, in *all* computational systems. Including the one in your head. When he said “distal access,” he assumed that you, too, have structures on one side of your head representing what you know about, say, Neil Armstrong, and a smaller structure on the other side of your brain encoding the *name* “Neil Armstrong,” and cognitive mechanisms allowing you to fetch the former given the latter.

True names, finance and prudent sorcerers

The BitTorrent example above is particularly interesting, since it comes with an unusual, distributed dereferencing mechanism. A BitTorrent name also has the additional exotic property of being a “true name,” in that it names *one and only one* piece of data. You don't need to trust the distributed store: every identifier includes enough information for you to verify that what your dereference produces is the thing originally named.

To invoke my opening quotation from the novel *True Names*, cultures have always attached a kind of magic to names, reflecting an intuitive understanding that names convey power and control; the reason for this is reflected in Newell's summary of names as being a *means of access*. According to the book of Genesis, for example, the first act of Man was the assignment of names [3], something which symbolically (there's that word, again) represents a transfer of control over the material world, as it is handed from its divine source over to human dominion.

Shifting from the sacred to the profane, these days I'm working with a bunch of finance people, who are so concerned with the issue that they devote really astounding amounts of brainpower and labor to keeping straight the names of things. An example will show why. Financial people not quite as obsessive about names as Jane Street's programmers have been known to use the name “TWTRQ” to purchase sizeable lots of Twitter stock. This is not a very good idea, because that actually gets you shares in the company Tweeter. Which is bankrupt. Twitter is traded on the stock exchange as “TWTR,” not “TWTRQ.” Likewise, Comcast has two different listings on the exchange (CMCSA and CMCSK), which get you different kinds of stock.

Oh, and the people who work on this stuff at Jane Street call what they do, “symbology.” They must have read Newell and Simon's Turing Award lecture, I guess.

Sharing and arrows

Another fundamental property I'd note that names have—besides ubiquity, and this notion of “distal access”—is that they give you the ability to refer to a thing *multiple times*. That's always a hint that maybe you need to be thinking about names. For example, if I have a computation represented by some expression ⟨*exp*⟩, and I want to do exactly one thing to the value it produces—say, I want to add 5 to it—then I simply write

```
⟨exp⟩ + 5
```

No need to name the value. But if I want to do *two* things to it, then I need to name it, so I can reference it:

```
let x = ⟨exp⟩ in
print(x); (* first use  *)
f(x+5);   (* second use *)
```

One way to view this is to say that a context-free grammar (like the one that says what strings are legal Java programs) turns a string of text into a tree structure, the parse tree. Once you put *names* into a language, however, your tree can now encode a DAG or (with recursive “letrec” scope) a general graph—names let you encode control- and environment-structure loops in your tree. When you need to write down something with DAG or cyclic structure, that's a hint you need to start thinking about some kind of a language with names in it.

So, I've now said the same thing a couple of times. Why not say it again? A name is an arrow: a link from arrow tail (reference) to arrow head (binding). (Or as compiler hackers prefer to say, from “use” to “def”) Some people take this arrow idea quite literally—for example, the Racket language's development environment will show you these links as actual arrows on your screen when you hover your mouse over a variable.

The art of names

Echoing my networking friend, I have a lot of respect for people who name well. Name choices inflict specific thought processes on people who use them; bad name choices inflict perverse or misleading thought processes, and make it hard to understand what's happening in a system. Good name choices make it easy and natural to do the right thing—like expressive, well-chosen types, they lead you effortlessly to the terms you wanted to write.

This is because names used by humans come with baggage. They fit into a larger framework. When you can get that framework right, and stick to it, then the names come easy—not just when they are minted, but when someone is trying to recall the name for some existing thing.

For example, when I'm using a library, and discover that

- you make new hash tables with `hashtable_create`,
- but new red/black trees with `make_rbtree`,
- and new skip lists with `NewSkipList`,

I cringe. Much, much better to fix on a single lexeme, such as “create”, and use that everywhere in the names of functions that make / create / allocate new things: `create_hashtable`, `create_rbtree`, and `create_skiplist`. Consistently constructing your names from a well-chosen set of such parts means that, once clients of your system have seen a couple of representative names, they can more or less *guess* the existence of functions they've never even seen, without having to paw through documentation or stop and look things up.

When I see someone agonising over what is just the right name for something he is defining, I relax a little bit: I know I'm working with someone who gets things right. Because naming things is one of the fundamental things *engineers* do.

The logic of names

One final remark about names. We use names all our lives, every day, all day. So they seem obvious and not so mysterious. But names are subtle. There is a sense in which the λ calculus is nothing more, really, than a theory of: names. Go look at the fundamental, definitional rules of the λ calculus. They are all about manipulating names! Consider, for example, β reduction, which slips some tricky renaming steps in behind the scenes as you proceed down into the redex. Likewise, the α rule tells you what it is about names that doesn't matter, and what it is about names that is of essence.

The λ calculus was a logician getting the handling of names right. It's amazing to me that this is such a recent step forward for the human race, something that has happened within living memory. And we've been working on names *formally*, not just in the street, for quite a while: a Greek formal-methods friend of mine [4] gives Aristotle credit for articulating the notion of “variables”—that is, names. When it takes over 2300 years to really nail down an idea, you figure the subject might be a little deeper than you initially supposed.

Another sign this is subtle is how many smart people got this wrong in the '70s and '80s by designing languages with dynamic scoping for their name handling. (Not that I'm, uh, naming any names.)

Yet another sign is that, in 2014, it's *still* a hot research topic! All the work on “macro hygiene” that comes out of the Scheme community? It's about names. The recent work done by Andy Pitts, Francois Pottier and others on nominal logics and nominal types? Nominal logic is just what the name says: a system whose entire *raison d'être* is manipulating names. Bob Harper and Derek Dreyer's work on module structures? Straightening out names as they are used in type classes and program modules [5].

Names have been significant in my own research life. For example, one of the most fun research results I ever had was an algorithm I developed with Mitch Wand that exploited a surprisingly simple data structure for representing the arrows that are names. I did some work in grad school on a family of program analyses that featured varying degrees of precision in how they abstracted *environment structure*: the key to the whole thing lay in the semantic mechanism that manages name spaces and name binding. Two of my top grad students have both done even more recent dissertations on novel, exciting mechanisms for reasoning about environments and the names they manage: Dimitrios Vardoulakis jumped the power of the abstraction up from finite to infinite domains, which is not so easy to do in a finite analysis; Matt Might's dissertation had no less than three distinct innovations concerning the design and management of abstract environments: abstract counting, abstract GC and frame-string contours.

But that's what's going on at the frontiers of scientific knowledge. Returning to the trenches of designing information systems and just doing my day-to-day programming: when I worry about handling names right, I tend to stop and bless the name of Church for getting things sorted for me, and am grateful I get to work in a tool for expression directly based on his results. As the man said of the λ calculus, “There may, indeed, be other applications of the system than its use as a logic.”

Well… yeah.
-Olin

Acknowledgements
Besides the names already mentioned in this essay, I'm also indebted to Harry Mairson and Alan Bawden for contributing to my ongoing education on the subject of names.

Footnotes

1 John Wroclawski

2 I actually had two thesis advisors: Peter Lee and Allen Newell. This is like winning the lottery two days consecutively.

3 Genesis 2:19 “And out of the ground the Lord God formed every beast of the field, and every fowl of the air; and brought them unto Adam to see what he would call them: and whatsoever Adam called every living creature, that was the name thereof.”

4 Panagiotis Manolios

5 …or so I am reliably assured by my friends who have the intellectual horsepower to understand their papers.