Structures

Ben Klemens

24 March 2011

First, let us go over the type system of every computing language in use today:

- There are basic types. These are numbers and words, in some form.
- There are lists or arrays of items, which have arbitrary length, and are indexed by a number.
- There are lists of items which are indexed by a name of some sort: dictionaries, hashes, or `structs`.

If you carry this list with you, it'll be easier to get a start in any new language. What are its basic types, how do we manipulate a list of identical types, and how do we aggergate diverse types into a structure of named elements?

[This is a lead-in for a longer series, by the way.]

Here's how the three types of type work out in practice among some popular languages:

	basic types	indexed lists	named lists
C family	int, float, char	arrays, pointers	structs
Lisp	numbers, strings	list	list
Awk	numbers, strings	—	array
Perl	$numbers, $strings	@array	%hash
Python	numbers, strings	list, tuple	dictionary
FORTRAN 77	int, float, char	arrays	—

This is off the top of my head—no need for irate letters about the things I approximated to keep the table short. But the organizational problem is the same in all cases: we sometimes have a bunch of homogeneous items, and we sometimes have disparate items, and we need a syntax for both types of organization.

There are a few quirks worth noting. Lisp is famous for using the same structure to handle both cases, which is all very neat and clean.

Awk is amusing in that it only has compound types indexed by a text name. You can fake normal indexed arrays with the naming scheme of `"1", "2", "3", ...`. Here's some sample code, set up so you can cut and paste it onto your command line:

```
echo '
BEGIN { try[0] = "zero";
        try[1] = 1;
```

```
        try[2] = 2;
        try["2"] = 8;
        for (i=0; i<=2; i++)
            print try[i];
} ' | awk -f '-'
```

You will see that `try[2]` will print `8`, meaning that the index is a string, even when you wrote it as a number. So that's nifty: if we have the dictionary/struct/hash, then we can fake simple arrays with them.

The table above counfounded two things: how we refer to an item (by number or a name), and whether the collection is homogeneous or of diverse types. For example, the awk code put both a string and integers into the same array, so awk's hashes are in the heterogeneous-holding category. For array-by-numeric-index, having homogeneous types is pretty much the norm, because if you're referring to a list of items that are only distinguished by which is first, second, third, ..., then they're probably pretty darn similar, type-wise (not to mention efficiency issues). Some languages do allow a list of differently-typed elements to be thrown into a list, so you could have `box[0]` be the box name, `box[1]` be a sublist of height/length/width, and `box[2]` be a pointer to the next box, but this is terrible form, and should be relegated to being an occasional lazy convenience. [I in fact did this in some code earlier today, so I can only be so righteous.]

I'll finish the thought next time, when I cover the named-index types: `structs` and dictionaries, and how these types are and are not similar across languages.