

Re: understanding arrays, and their use

Source: <http://coding.derkeiler.com/Archive/Tcl/comp.lang.tcl/2005-06/msg00704.html>

- *From:* Darren New <dnew@xxxxxxxxxx>
 - *Date:* Thu, 23 Jun 2005 21:50:45 GMT
-

Mr.B wrote:

I am having a terrible time trying to comprehend arrays.

Arrays are actually very simple. They're hashtables mapping strings to other strings.

There are several dozen lines in my real data, but it shouldn't make much difference. There are extra spaces between the fields - I want to treat more than one space the same as one space. Basically there are 4 fields. The fourth field can be a string enclosed in quotes with either a file path, or multiple words separated by spaces. I need to treat the 4th group as one field. I want to pull everything "toola" out of the file, into an array. Later, I want to go through the array, and set a variable named for each option (option1, option2, etc) to the value in field 4.

You would be well-served to rephrase this in terms of the steps you want to take, rather than disconnected requirements. Tcl is procedural, so simply listing the requirements without putting them together as a procedure will make the task harder to understand.

I haven't tested this code, but it should be easy to fix bugs.

You want to read the file a line at a time. You want to break it into space-separated groups, with four groups per line. The fourth group could have quotes around it, or not. If the first item in the group matches the thing you're looking for, the second item should be matched to the fourth item in an array.

So, reading a file a line at a time...

```
set filehandle [open myinput.txt r]
while {-1 != [gets $filehandle line]} {
```

Re: understanding arrays, and their use

```
    ...
}
close $filehandle
```

(BTW, if you use [read], note the "--nonewline" option so you can split it without getting an extraneous blank line.)

Now, separate a line into four groups, with whitespace separating them. Probably a regular expression will do the trick most easily, altho it's often easier to use less complex solutions. I haven't tested this regular expression, and there's probably a more compact way of writing "whitespace" and "not whitespace", and you could probably handle the quote marks right in the regexp if you were intimately familiar with the assignment-to-result-variables rules, but ...

```
set e {^[ \t]*([^\t+)[ \t]+([^\t+)[ \t]+([^\t+)[ \t]+(.[\t]*$}
set filehandle [open myinput.txt r]
while {-1 != [gets $filehandle line]} {
    set matched [regexp $e $line junk one two three four]
    ... do something with one two three four ...
}
close $filehandle
```

Now, what do you want to do? If "one" is what you're looking for, then put "two" and "four" together. Take the quotes off "four" first. This assumes there aren't embedded quotes in four, i.e., that there is either zero or one quote at either end. It also skips lines that it can't match, for example having only 3 fields.

```
set tool "toola"
set e {^[ \t]*([^\t+)[ \t]+([^\t+)[ \t]+([^\t+)[ \t]+(.[\t]*$}
set filehandle [open myinput.txt r]
while {-1 != [gets $filehandle line]} {
    set matched [regexp $e $line junk one two three four]
    if ($matched && $tool == $one) {
        set lookup($two) [string trim $four \"]
    }
}
close $filehandle
parray lookup ; # this prints your result.
```

If you really want separate variables holding the options, you can then say
foreach inx [array names lookup] {set \$inx \$lookup(\$inx)}
but I suspect you don't need to do that.

Note that you'll need to unset "lookup" before calling this a second time (in order to get rid of options for toola that aren't in toolb), and that if you want to return the value of "lookup" from a proc, you'll need to

Re: understanding arrays, and their use

understand how to do that.

```
set lookup($first) [list $second]
```

The reason you got strange answers is you never put the fourth value anywhere.

--

Darren New / San Diego, CA, USA (PST)

The samba was clearly inspired
by the margarita.

.