Notes on tutorial #2 (for week 5)
(Feb 6, 2004)

17th February 2004

Q1: The claim is made that **`Tcl`** uses *associative* indexes into arrays. Give an example (in **`Tcl`**) of a useful use of the *associative* array, comparing it with similar code for a *normal* array.

Answer: *The indexes can be anything - including text. They can be used to represent 1 and N-dimensional arrays, sequential structures like lists and queues (though Tcl provides these natively), symbol tables, Cartesian products (i.e., the structs of C, the records of Pascal), graphs, sets, and more.. (from Tcl notes at uchicago).*

```
set salary(Smith) 30000
set age(Smith) 45
set length($s) [string length $s]
```

Q2: Try running **`tclsh`** on one of the UNIX systems. **`Tcl`** does not have the commands **`ls`** or **`ps`** or **`emacs`**, but try typing them anyway.

(a) Explain the results of this experiment.

Answer: *The corresponding UNIX commands run. If a command is not found in the list of* **`Tcl`** *commands, then it searches for a UNIX command of that name.*

(b) Explain also why this behaviour of **`tclsh`** may result in non-portable programs.

Answer: *The UNIX commands are different from the DOS/Windows commands. If a program internally used a UNIX command, then it would not be portable to a Windows machine. Note that when running a* **`Tcl`** *script, you will have to do it using something like:*

```
puts [exec /bin/ps]
```

Q3: Differentiate between a self-interpreter, and a language like **`Tcl`** with the self-interpretation property.

Answer: *A self interpreter is a program written in a language $\mathcal{L}$ which interpets the language $\mathcal{L}$. Most high level languages can have a self interpreter written, although it may be large.*
The self-interpretation property found in **`Tcl`** *allows you to easily interpret an arbitrary string as a* **`Tcl`** *program (without having to write a self-interpreter).*

Q4: Modify the **`Tcl/Tk`** "Hello World" program to print out the date and time on the console.

(a) Demonstrate your new program

Answer: *Something like the following should be OK.*

```
#!/usr/local/bin/wish
button .quit -text "Hello" -command {puts [exec /bin/date]}
pack .quit
```

(b) In your program you are probably using **`-command {....}`**. Explain what happens when you replace **`{....}`** with **`"...."`**.

Answer: *I had* **`{puts [exec /bin/date]}`**. *Replacing the* **`{`** *with* **`"`** *does not escape the operation of the* **`[`** *substitution, and so we end up with a command like:*

```
puts Fri Feb 6 12:12:24 GMT-8 2004
```

leading to an error message like:

```
wrong # args: should be "puts ?-nonewline? ?channelId? string"
```

Q5: Write Tcl code to:

(a) write the **`"`** character to the console output

Answer: *Simple - just use backslash substitution:*

```
puts \"
```

(b) write the **`{`** character to the console output

Answer: *Simple - just use backslash substitution:*

```
puts \{
```

(c) write out program arguments (if you call a tcl program: **`myprog a b c`**, it will display **`a b c`**)

Answer: *There are two variables argc (the count), and argv - a list of the arguments.*

```
puts $argv
puts [lindex $argv 1]
```

(d) safely open a text file and print its contents. If the file is not openable for some reason, print out a suitable error message.

Answer: *Something like this:*

```
if [catch {open [lindex $argv 0] r} fileId] {
    puts stderr "Cannot open [lindex $argv 0]: $fileId"
} else {
    puts [read $fileId]
    close $fileId
}
```