Tcl/TK Tutorial

Learning Tcl/TK

- What is Tcl/TK?
  - An interpreted programming language
    - Build on-the-fly commands, procedures
    - Platform-independent
    - Easy to use for building GUIs
- Need little experience with programming
  - Easy
  - Programs are short, efficient
- Be willing to learn something new

Why Tcl/TK?

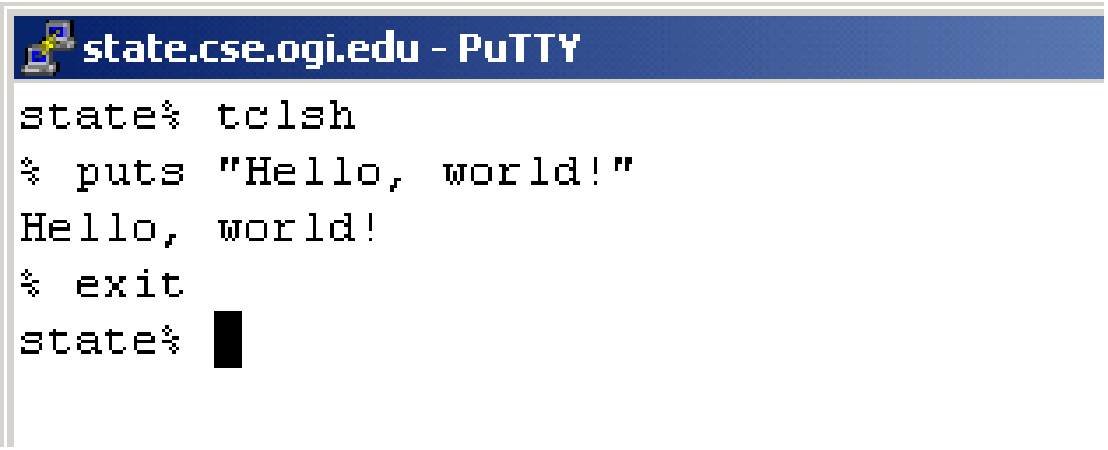
- Easy, fast programming
- Free
- Download & install Tcl/TK 8.4 on your own
  - CSE machines (state) are set up with Tcl/TK 8.0
  - http://tcl.activestate.com/software/tcltk/downloadnow84.tml
- Lots of online documentation, mostly free
- Solutions for AI homework will be in Tcl
- Base for the CSLU toolkit

Hello World

- How to run your Tcl program
  - Command line (state.cse.ogi.edu or DOS)
    - Type "tclsh" to launch the console
  - Type your program directly on the console
  - Use the command "source" (source filename)
  - Double click your .tcl file (if associated)
- Output on the console
  - Command: `puts "Hello, world!"`

Hello World

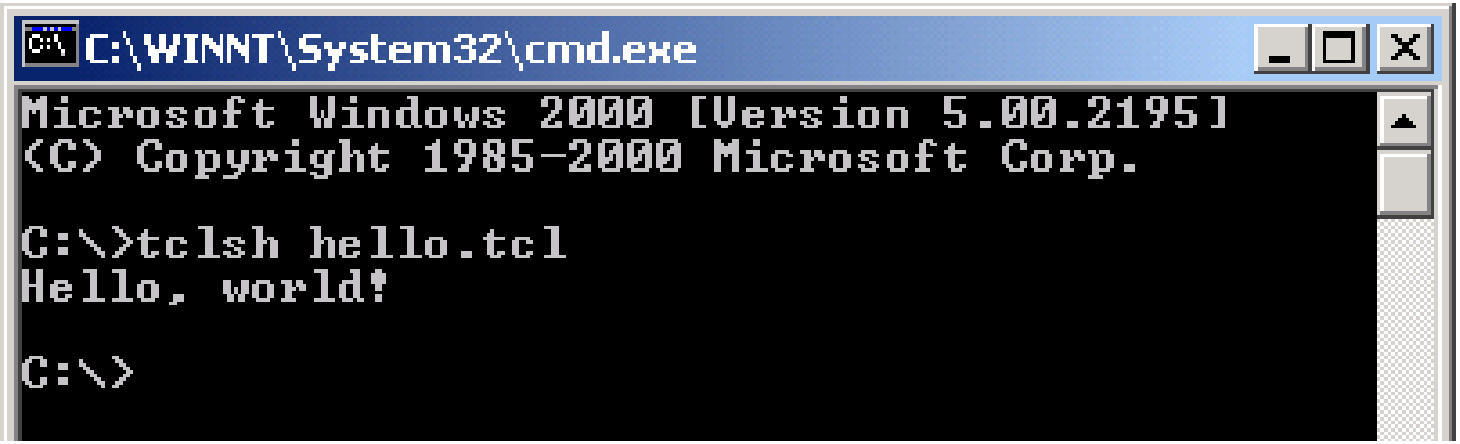
- Command line (state.cse.ogi.edu or DOS)
  - Type "tclsh" to launch the console
  - Type tcl code into console



Hello World

- Sourced on the console
  - Type "tclsh", followed by name of program file

```
######### hello.tcl ########
puts "Hello, world!"
```



Hello World

- Double-clicking your .tcl file (if associated with wish84.exe)

```
######## hello.tcl #########
Hello.tcl
  wm withdraw .
  console show
  puts "Hello, world!"
```

Basic operations

- print to screen (`puts`)

```
puts –nonewline "Hello, world!"
puts "!!"
```

- assignment (`set`)

```
set income 32000
puts "income is $income"
```

(using '$' to get the value of a variable)

- mathematical expressions (`expr`)

```
set a 10.0
expr $a + 5
expr int($a/3)
```

Some useful commands

- unset: destroy a variable

```
unset num
```

- info: check whether the named variable has been defined

```
if {![info exists num]} {
    set num 0
}
incr num
```

- window commands

```
wm withdraw .
console show
```

Special characters

`#` : single-line comments, similar to "//" in C

`;#` : in-line comments, just like "//" in C

`\` : escape character, same function as in C

: also used to break a long line of code to two lines

`$` : get the value of a variable

- var : name of variable
- $var : value of variable

`[]` : evaluate command inside brackets

Control structures (1)

- if then else

```
set income 32000
if {$income > 30000} {
    puts "$income -- high"
} elseif {$income > 20000} {
    puts "$income -- middle"
} else {
    puts "$income -- low"
}
```

- while loops

```
set i 0
while {$i < 100} {
    puts "I am at count $i"
    incr i
}
```

Control structures (2)

- for loops

```
for {set i 0} {$i < 100} {incr i} {
    puts "I am at count $i and going up"
    after 300
    update
}
for {set i 100} {$i > 0} {set i [expr $i - 1]} {
    puts "I am at count $i and going down"
}
```

- foreach loops

```
set lstColors {red orange yellow green blue purple}
foreach c $lstColors {
    puts $c
}
```

Control structures (3)

- foreach loops (con't)

```
set lstColors {red orange yellow green blue purple}
foreach {a b c} $lstColors {
    puts "$c--$b--$a"
}

set lstFoods {apple orange banana lime berry grape}
foreach f $lstFoods c $lstColors {
    puts "a $f is usually $c"
}

foreach {a b} $lstFoods c $lstColors {
    puts "$a & $b are foods. $c is a color."
}
```

Procedures

- procedure calls (embedded commands)

```
set b [expr $a + 5]
puts "The value of b is $b"
```

- create your own procedure (called by value only)

```
proc foo {a b c} {
    return [expr $a * $b - $c]
}
puts [expr [foo 2 3 4] + 5]

proc bar { } {
    puts "I'm in the bar procedure"
}
bar
```

Variable scope

local and global variables

```
set a 5
set b 6
set c 7
proc var_scope { } {
    global a
    set a 3
    set b 2
    set ::c 1
}
var_scope
puts "The value for a b c is: $a $b $c"
```

Lists in Tcl/TK

- Everything is a list!
- Many ways to create a list

```
set myList [list a b c]
set myList "a b c"
set myList {a b c}

set myList [list $a $b $c]
set myList {$a $b $c}

set myList [list a   b   c]
set myList "a   b   c"

set s Hello
puts "The length of $s is [string length $s]."
=> The length of Hello is 5.
puts {The length of $s is [string length $s].}
=> The length of $s is [string length $s].
```

List operations

```
set lstStudents [list "Fan" "Kristy" "Susan"]
puts [lindex $lstStudents 0]
puts [lindex $lstStudents end]
puts [llength lstStudents]  (unexpected result!)
puts [llength $lstStudents]
lappend $lstStudents "Peter"  (wrong!)
lappend lstStudents "Peter"
puts [linsert lstStudents 2 "Tom"]  (wrong!)
puts [linsert $lstStudents 2 "Tom"]
set lstStudents [linsert $lstStudents 2 "Tom"]
set lstStudents [lreplace $lstStudents 3 3 "Rachel"]
set lstStudents [lreplace $lstStudents end end]
set lstStudents [lsort –ascii $lstStudents]
puts [lsearch $lstStudents "Peter"]
```

Lists of lists (of lists…)

```
set a [list [list x y z]]
puts [lindex $a 0]
puts [lindex [lindex $a 0] 1]
puts [lindex [lindex $a 1] 0]
```
(unexpected result)

```
set a [list x [list [list y] [list z]]]
=> How to get to the z?

set arg1 [list g [list f [list h [list i X]]] [list r Y] k]
set arg2 [list g [list f [list h [list i Y]]] [list r b] L]
set both [list $arg1 $arg2]
puts $both
```

Array operations

Associative arrays (string as index)

```
set color(rose) red
set color(sky) blue
set color(medal) gold
set color(leaves) green
set color(blackboard) black
puts [array exists color]
```

(tests if an array with the name "color" exists)

```
puts [array exists colour]
puts [array names color]
```

(returns a list of the index strings)

```
foreach item [array names color] {
    puts "$item is $color($item)"
}
```

(iterating through array)

```
set lstColor [array get color]
array set color $lstColor
```

(convert array to list)

(convert list to array)

Regular expressions

- regsub

```
set stmt "Fan is one of Shania's fans"
regsub –nocase "fan" $stmt "Kristy" newStmt
```

?*switches*? *exp* *string* *subSpec* ?*varName*?

```
puts "$newStmt"
regsub –nocase –all "fan" $stmt "Kristy" newStmt
puts "$newStmt"
```

- regexp

(returns 1 if the regular expression matches the string, else returns 0)

```
puts [regexp –nocase "fan" $stmt]
```

?switches? *regexp* *string*

- format

```
puts [format "%s is a %d-year-old" Fan 26]
```

formatString ?*arg arg* ...?

String operations

```
set statement "    Fan is a student    "
set statement [string trim $statement]
puts [string length $statement]
puts [string length statement]
puts [string index $statement 4]
puts [string index $statement end]
puts [string first "is" $statement]
     (string last)
puts [string first $statement "is"]
puts [string range $statement 4 end]
puts [string replace $statement 9 end "professor"]
puts [string match "*student" $statement] (* ? [])
```

File operations

```
set fRead [open source.txt r]
set fWrite [open target.txt w]
while {![eof $fRead]} {
  set strLine [gets $fRead] ;#or gets $fRead strLine
  regsub –nocase –all "fan" $strLine "kristy" strLine
  puts $fWrite $strLine
}
close $fRead
close $fWrite
```

################## source.txt ##################

Fan is a CSE student.

Fan is also one of Shania’s fans.

Kristy and Fan are classmates.

Miscellaneous commands

- **eval**: execute a command dynamically built up in your program

```
set Script {
    set Number1 17
    set Number2 25
    set Result [expr $Number1 + $Number2]
}
eval $Script
```

- **exec**: execute external programs
- **clock**

Debugging your program

- Use `puts` statements (with `update` and `after` when using wish84.exe to run program)
- tk_messageBox: pop up a message box

```
tk_messageBox –message "run to here" –type ok
```

- tclpro
- **trace variable** variableName operation procedure

Common pitfalls

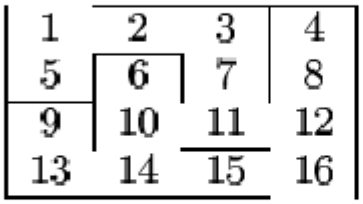
- Missing `$` or extraneous `$`
- Using `{a}` vs `"a"` vs `[list a]`
- Creating list items that are empty lists

```
a b {} d
```

Maze Tcl example

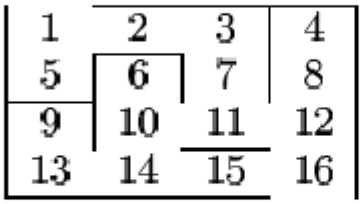
pseudocode:

```
create a path which just has the start state
make this path the only member of the list of alternatives to be explored
while list of alternatives is not empty and not done
   set firstpath to be the first path from the list of alternatives
   update alternatives so it doesn't include the first path
   set last to be the last member of firstpath
   for each cell connected to the last member
      create newpath with cell at the end of firstpath
      if cell is 16
         display path
      else
         add newpath to end of list of alternatives
```



Maze Tcl example

```
set bDone 0
set path [list 1]
set alternatives [list $path]
while {[llength $alternatives] > 0 && !$bDone} {
    set firstpath [lindex $alternatives 0]
    set alternatives [lrange $alternatives 1 end]
    set last [lindex $firstpath end]
    foreach cell $connected($last) {
        set newpath [linsert $firstpath end $cell]
        if {$cell == 16} {
            puts "Answer is $newpath"
            set bDone 1
            break
            update
            after 1000
        } else {
            lappend alternatives $newpath
        }
    }
}
```



Tcl references and resources

- Help file
- http://www.tcl.tk/scripting/
- http://www.msen.com/~clif/TclTutor.html
- Search site:
  http://xyresix.com/nwsbook/search.html
- List of Tcl commands with man pages
  http://tcl.activestate.com/man/tcl8.4/TclCmd/contents.htm
- Tcl examples:
  http://www.beedub.com/book/2nd/tclintro.doc.html#2668
- All code in this tutorial (plus some) and expected output
  ftp://cslu.ece.ogi.edu/pub/kristina/
  tutorial.tcl, tutorial_output.txt, source.txt

Tcl editors

- emacs
- TextPad
  - http://www.textpad.com/
  - http://www.textpad.com/add-ons/synn2t.html - TCL/TK (5)

Reminder

- Tuesday's classes are still from 11:30-12:50