UNIVERSITY OF WATERLOO
Department of Electrical and Computer Engineering

An Introduction To Tcl/Tk

Christopher Trudeau
`ctrudeau@etude.uwaterloo.ca`

Parallel and Distributed Systems Group

slides by *SlickSlides*

Scripting Basics

- interpreted executable text files
- first line indicates what interpreter to use:
  - #!/usr/bin/sh
  - #!/.software/local/.admin/bins/bin/tclsh
  - #!/.software/local/.admin/bins/bin/wish -f
- typically typeless (single data type)
- most use string formats for storage
- tcl, *tool command language*, is a scripting language
- tk, *toolkit*, is an X Motif-like extension to tcl
- tcl/tk created by John K. Ousterhout

Tcl/Tk Operators

- case sensitive
- comments defined using #, must be on own line!
- multiple commands on a line using ;
- lines can be continued using \
- basic C operators:

  `+ - * / ! ~ << >> < > <= >= == != & ^ && || ?:`
- variables begin with a letter, can contain letters, numbers and underscores
- variable value returned using $ operator
- examples of numbers: `0123, 0x3F, 2.1, 7.1e+16 6E4`
- standard C order of operations
- assignment operator: `set` function

```
set num 3
puts $num
unset oldnum
```

- no ++ or `--`, uses `incr` function

```
incr num
incr num 4
```

Tcl Substitution, Expressions & Evaluation

- variable substitution is done on all strings *once* by the interpreter
- can stop substitution with \ or {}

```
puts -nonewline "Hello there num $num"
puts "I just printed the contents of \$num"
```

- function `expr` returns the value of an expression
- evaluation of a string, as a script, is done with []

```
set b [expr $a + 5]
```

- associative arrays defined using ()

```
set month(jan) 0
set matrix(1,3) 5
```

- lists are strings with items separated by spaces
- lists can be manipulated using `lappend`, `lindex`, `list`, `llength`, `lrange`, `lreplace`, `lsearch`, `linsert`, `concat`
- `concat` puts all arguments into a single list
  - `concat {a b c} d` ⇒ "a b c d"
- `list` uses each argument as a separate element in a list
  - `list {a b c} d` ⇒ {a b c} d

Tcl Functions

- functions defined using the keyword `proc`
- parameters enclosed in {}
- keyword `global` allows access to global variables
- keyword `return` returns a values from the function

```
proc PrintInfo {name number} {
   global company

   puts $company $name $number
   return [expr $number + 1]
}
```

- variable parameters accessed by using keyword `args`

```
proc PrintThird args {
   puts [lindex $args 2]
}
```

Tcl Strings & Regular Expressions

- function called `string` allows you to manipulate strings
- `string` takes a parameter indicating what to do
- parameters are `compare`, `first`, `index`, `last`, `length`, `match`, `range`, `tolower`, `toupper`, `trim`, `trimleft`, and `trimright`

```
string first t "There is the picture tube"
# returns 9 -- because of the first "t"
string length "sample string"
# returns 13
string trim aaxxxbab abc
# returns "xxx" as we are trimming a, b, and c from both ends
string match Tcl* "Tcl is cool"
# returns 1 as glob matching succeeded
string range "My funky string" 3 7
# returns "funky"
```

- regular expressions can be evaluated using `regexpr` and `regsub`

```
regexp -nocase [a-z] A
# returns 1
regexp {([0-9]+) *([a-z]+)} "Walk 10 km" a b c
# a gets match: "10 km", b & c get portions "10" and "km"
# also returns 1 to indicate a match
regsub there "They live there lives" their x
# x gets "They live their lives"
# returns 1 to indicate substitution
```

Tcl Control Flow

- `if` statement similar to C

```
if { $i == 3 } {
   puts "got here"
} elseif { $i == 4 } {
   puts "or here"
} else {
   puts "or even here!"
}
```

- `while` statement is similar to C

```
while { $i < 3 } {
   puts $i
   incr i
}
```

- `for` statement is... yep, similar to C

```
for { set i 0 } { $i < 3 } { incr i } {
   puts $i
}
```

- `foreach` statement

```
foreach i "one two three" { puts $i }
```

- `switch` is similar to C

```
switch $x {
   a -
   b { incr t1 }
   c { incr t1 3 }
}
```

Tk and Widgets

- tk based on X widgets
- create widgets with functions: `button`, `label`, `entry`, `frame`, `listbox`, `checkbutton`, `radiobutton`, `scale`, `menu`
- first parameter is the name of the widget to create
- name must begin with a `.` and a lower case letter

```
button .bPushMe
```

- widgets support a common set of options: `-text`, `-fg`, `-bg`, `-width`, `-height`, `-borderwidth`, `-relief`, `-anchor`
- a declared widget name is also a function for accessing that widget

```
.bPushMe configure -fg Red
```

- creation of a widget does not place it on the screen

```
pack .bPushMe
```

- distances are in pixels or specified with `c`, `i`, `m`, `p` for centimetres, inches, millimetres, and points

```
frame .fMyFrame -width 3c -height 30m -borderwidth 4
```

Tk Geometry Manager

- default placement is each widget on top of another

```
#!/.software/local/.admin/bins/bin/wish -f
wm title . "Button"

button .b1 -bg white -relief raised -text raised
button .b2 -fg #F0F -relief sunken -text sunken
button .b3 -fg #FF0000 -relief flat -text flat
button .b4 -fg #FFF000FFF -relief groove -text groove
button .b5 -fg #FFFF0000FFFF -relief ridge -text ridge -command exit
pack .b1 .b2 .b3 .b4 .b5
```

- use `-side` option with `left, right top, bottom` for placement

```
label .l1 -text "Enter info:"
entry .e1 -textvariable info
button .b1 -text "Exit" -command exit
pack .l1 .e1 .b1 -side left
```

Tk Advanced Placement

- use `frame` widgets and the `-in` option

```
frame .fMenu
   menubutton .fMenu.file -text "File" -menu .fMenu.file.m
   menu .fMenu.file.m -tearoff 0
   .fMenu.file.m add command -label "Open"
   pack .fMenu.file -in .fMenu -side left
   button .bExit -text Exit -relief raised -command exit
   pack .bExit -in .fMenu -side right
pack .fMenu -fill x

frame .fContain
frame .fEmpty -width 3c
frame .fButtons
for {set row 1} { $row < 4 } { incr row } {
   frame .fRow($row)
   for {set col 1} { $col < 4 } { incr col } {
      button .b($row,$col) -relief raised -text "$row.$col"
      pack .b($row,$col) -in .fRow($row) -side left
   }
   pack .fRow($row) -in .fButtons
}
pack .fEmpty .fButtons -side left -in .fContain
pack .fContain
```