Today's Lecture

1. Assignment II

2. `Tcl/Tk`
   - `Tcl` is a command interpreter (like a shell)
   - `Tk` is a library of graphical widgets (that `tcl` can use)

3. Interactive demos

`tcl`: Interpreted Programming Language


```
$ /usr/princeton/bin/tclsh
%
```

- Interactive or scripted
- Extremely dynamic
- Variables and substitutions
- Control structures & procedures
- Datatypes: integers, reals, strings
  (but most everything is internally a string)
- List datatype
- Error handling: exceptions
- Files, processes, ...

`tcl`: Variables

`set` and `unset` respectively introduce and remove variables

`$`x fetches the contents of variable x

one-dimensional arrays; multi-dimensional arrays can be "faked"

array sizes need not be declared

arrays are *associative*; subscripts may be strings

`tcl`: Substitutions and Quoting

{ braces are the primary quoting mechanism }

Arithmetic expressions introduced by `expr`;
for example, `expr $i-1`

Evaluation of a term is requested by placing it in square brackets: `[puts "hello"]`

`eval` can be used to force further evaluation:

```
% set cmd "set a 0"
set a 0
% eval $cmd
0
% puts $a
0
%
```

- `if`
- `while`
  may `break` and `continue` loop
- `for`
- `foreach`
- `switch`
  labels may be regular expressions
- `proc`
  use `return` to return a result

`tk` is a library of *widgets* for building user interfaces

`wish` is a `tcl` shell that contains the `tk` widgets:

```
$ /usr/princeton/bin/wish
%
```

(this also creates a window—the *main* widget named "`.`")

A user interface consists of a *hierarchy* of widgets

"Dewey decimal" naming of widgets: *e.g.*, `.graph.xscroll`

When an action occurs in a widget (*e.g.*, left mouse-button press), `tk` is given an *event* (*e.g.*, `<Button-1>`)

Code can be *bound* to a widget; this code executes when a certain event occurs

Some Widgets:

- Frames (`frame`)
- Labels (`label`)
- Canvases (`canvas`)
- Buttons (`button`, `checkbutton`, `radiobutton`)
- Messages (`message`)
- Listboxes (`listbox`)
- Scrollbars (`scrollbar`)
- Scales (`scale`)
- Menus (`menu`, `menubutton`)

Widget geometry managed by a *packer*

```
pack .aFrame .aButton .aCanvas -side left
```



```
#!/usr/bin/tclsh
# tgrep.tcl

if {$argc != 2} {
    error "usage: $argv0 pattern filename"
}

set f [open [lindex $argv 1] r]
set pattern [lindex $argv 0]
while {[gets $f line] >= 0} {
    if [regexp $pattern $line] {
        puts $line
    }
}
close $f

# end of tgrep.tcl
```

```
#!/usr/bin/tclsh
# array.tcl

set ExampleOf(Dog) Labrador
set ExampleOf(Cat) Tiger
set ExampleOf(Fish) Guppy

while {![eof stdin]} {
        puts -nonewline ">"
        set line [gets stdin]
        if {$line == ""} {
            continue;
        }
        if [catch {set result $ExampleOf($line)}] {
                puts "No example for: $line"
        } else {
            puts $result
        }
}

# end of array.tcl
```



```
# watch.wish

set instance 0

proc watch name {
        global instance

        toplevel .watch_$instance
        label .watch_$instance.label -text "Value of \"$name\": "
        label .watch_$instance.value -textvariable $name
        pack .watch_$instance.label .watch_$instance.value -side left
set instance [expr $instance+1]
}

# end of watch.wish
```



```
#!/usr/bin/wish -f
# browse.wish (from tk demo library)

#
# This script generates a directory browser, which lists the working
# directory and allows you to open files or subdirectories by
# double-clicking.

# Create a scrollbar on the right side of the main window and a listbox
# on the left side.

scrollbar .yscroll -command ".list yview"
pack .yscroll -side right -fill y
listbox .list -yscroll ".yscroll set" -relief raised -geometry 15x20 \
        -setgrid yes
pack .list -side left -fill both -expand yes
wm minsize . 1 1
```



```
# The procedure below is invoked to open a browser on a given file;  if the
# file is a directory then another instance of this program is invoked; if
# the file is a regular file then the Mx editor is invoked to display
# the file.

proc browse {dir file} {
    global env
    if {[string compare $dir "."] != 0} {set file $dir/$file}
    if [file isdirectory $file] {
        exec browse.wish $file &
    } else {
        if [file isfile $file] {
            if [info exists env(EDITOR)] {
                eval exec $env(EDITOR) $file &
            } else {
                exec xedit $file &
            }
        } else {
            puts stdout "\"$file\" isn't a directory or regular file"
        }
    }
}
```

```
# Fill the listbox with a list of all the files in the directory (run
# the "ls" command to get that information).

if $argc>0 {set dir [lindex $argv 0]} else {set dir "."}
foreach i [exec ls -a $dir] {
    .list insert end $i
}

# Set up bindings for the browser.

bind .list <Control-q> {destroy .}
bind .list <Control-c> {destroy .}
focus .list
bind .list <Double-Button-1> {foreach i [selection get] {browse $dir $i}}

# end of browse.wish
```



```
#!/usr/bin/wish -f
# graph.wish  (from Ousterhout's Tcl/Tk book)

canvas .c
pack .c

proc mkNode {x y} {
        global nodeX nodeY edgeFirst edgeSecond
        set new [.c create oval [expr $x-10] [expr $y-10] \
                 [expr $x+10] [expr $y+10] -outline black \
                 -fill white -tags node]
        set nodeX($new) $x
        set nodeY($new) $y
        set edgeFirst($new) {}
        set edgeSecond($new) {}
}

proc mkEdge {first second} {
        global nodeX nodeY edgeFirst edgeSecond
        set edge [.c create line $nodeX($first) $nodeY($first) \
                  $nodeX($second) $nodeY($second)]
        .c lower $edge
        lappend edgeFirst($first) $edge
        lappend edgeSecond($second) $edge
}
```



```
bind .c <Button-1> {mkNode %x %y}
.c bind node <Any-Enter> {
        .c itemconfigure current -fill white
}

bind .c 1 {set firstNode [.c find withtag current]}
bind .c 2 {
        set curNode [.c find withtag current]
        if {($firstNode != "") && ($curNode != "")} {
                mkEdge $firstNode $curNode
        }
}
focus .c

# end of graph.wish
```


Computers are still too slow to write this in an interpreted language like `tcl`!

```
#!/usr/bin/wish -f
# sand.wish  -- simulate falling grains of sand, sort of

canvas .c
set grainWidth 10
set szX 20
set szY 20

proc random {} {
        return [gets stdin]
}

proc show {} {
        global lights grid szX szY
        for {set i 0} {$i < $szX} {incr i} {
                for {set j 0} {$j < $szY} {incr j} {
                        if {$grid($i,$j)} {
                                .c itemconfigure $lights($i,$j) -fill blue
                        } else {
                                .c itemconfigure $lights($i,$j) -fill black
                        }
                }
        }
        # flush events
        update
}
```

```
# a crude model for moving a piece of sand
proc move {i j} {
	global grid szX

	set below [expr $j+1]
	if {($grid($i,$j))} {
		if {!($grid($i,$below))} {
			set grid($i,$below) 1
			set grid($i,$j) 0
		} elseif {($i > 0) && !($grid([expr $i-1],$below))} {
			if {($i < [expr $szX-1]) && \
			    !($grid([expr $i+1],$below))} {
				set grid($i,$j) 0
				if [random] {
					set grid([expr $i-1],$below) 1
				} else {
					set grid([expr $i+1],$below) 1
				}
			} elseif [random] {
				set grid([expr $i-1],$below) 1
				set grid($i,$j) 0
			}
		} elseif {($i<[expr $szX-1]) && \
			  !($grid([expr $i+1],$below)) && \
			  [random]} {
			set grid([expr $i+1],$below) 1
			set grid($i,$j) 0
		}
	}
}
```



```
proc relocate {} {
	global szX szY

	for {set i 0} {$i < $szX} {incr i} {
		for {set j [expr $szY-2]} {$j >= 0} {incr j -1} {
			move $i $j
		}
	}
}

proc init {} {
	global lights grid grainWidth szX szY
	for {set i 0} {$i < $szX} {incr i} {
		for {set j 0} {$j < $szY} {incr j} {
			set lights($i,$j) \
				[.c create rectangle \
				 [expr $i*$grainWidth] \
				 [expr $j*$grainWidth] \
				 [expr $i*$grainWidth+$grainWidth-1] \
				 [expr $j*$grainWidth+$grainWidth-1] \
				 -fill blue -width 0]
		}
	}
}
```



```
proc setgrid {f} {
	global grid szX szY

	for {set i 0} {$i < $szX} {incr i} {
		for {set j 0} {$j < $szY} {incr j} {
			set grid($i,$j) [eval $f]
		}
	}
	show
}


proc mkRain {} {
	global grid szX

	if [random] {
		set grid([expr ($szX/2)+[random]-[random]],0) 1
	}
}

proc doit {} {
	global isRaining

	while {1} {
		relocate
		if {$isRaining} {
			mkRain
		}
		show
	}
}
```



```
# set up the control panel
toplevel .cntrls
wm title .cntrls "Controls"
button .cntrls.quit -text Quit -command exit
button .cntrls.doit -text Doit -command doit
button .cntrls.randomize -text Randomize -command "setgrid random"
checkbutton .cntrls.rain -text Rain -variable isRaining

pack .cntrls.rain .cntrls.randomize .cntrls.doit .cntrls.quit -side left

pack .c
focus .c

# startup
init
setgrid "expr 0"
show
doit

# end of sand.wish
```

Another graphical shell: Visual Basic