TCL/TK
Scripting Language

Computer Laboratory - ICT1232

W.M.P.N. Jayaweera
TG/2018/392
Department of ICT
Faculty of Technology
University of Ruhuna

Table of Contents

What is TCL

Tcl is a high-level, interpreted, general purpose multi-paradigm* programming language. The name originally comes from **T**ool **C**ommand **L**anguage, but is conventionally spelled "Tcl" rather than "TCL".

Programming paradigms are a way to classify programming languages based on their features. Languages can be classified into multiple paradigms.

Common programming paradigms include:

- *Imperative in which the programmer instructs the machine how to change its state,*
    - *Procedural which groups instructions into procedures,*
    - *Object-oriented which groups instructions together with the part of the state they operate on,*
- *Declarative in which the programmer merely declares properties of the desired result, but not how to compute it*
    - *Functional in which the desired result is declared as the value of a series of function applications,*
    - *Logic in which the desired result is declared as the answer to a question about a system of facts and rules,*
    - *Mathematical in which the desired result is declared as the solution of an optimization problem*

It is commonly used embedded into C applications, for rapid prototyping, scripted applications, GUIs, and testing. TCL interpreters are available for many operating systems, allowing TCL code to run on a wide variety of systems. Because TCL is a very compact language, it is used on embedded systems platforms, both in its full form and in several other small-footprint versions.

TCL is a scripting language that aims at providing the ability for applications to communicate with each other. On the other hand, Tk is a cross platform widget toolkit used for building GUI in many languages.

How to execute TCL

First, you have to install it to your system. If your Linux system is based on Debian (like Ubuntu) you can install it by executing the following command. (But most of the systems have pre installed TCL)

```
$ sudo apt-get install tk8.5 tcl8.5
```

You can save your TCL source as .tcl file. Then you can execute it as follows. (Assume your file name is HelloWorld.tcl)

```
$ tclsh HelloWorld.tcl
```

TCL scripts

TCL scripts should have have inserted `#!/usr/bin/tclsh` header to your TCL file, then you can run it using `./HelloWorld.tcl` after giving execute permissions.

TCL Substitution type

There are three kinds of substitutions in TCL

1. Command substitution
2. Variable substitution
3. Backslash substitution

Command substitution

Square brackets are used for command substitution. "expr" used for performing the arithmetic calculation.

Eg:

```
#!/usr/bin/tclsh
puts "5 x 2 = [expr 5*2]"
```

OUTPUT =

```
5 x 2 = 10
```

Variable substitution

TCL performs variable substitution with the help of $ sign.

Eg:

```
#!/usr/bin/tclsh
set  Year  2020
puts  Year
puts  $Year
```

OUTPUT =

```
Year
2020
```

It prints variable value only there is a $ sign in front of the variable name. If not, it simply prints the variable name as a string value.

Backslash substitution

In Tcl, the backslash is used for escaping special characters as well as for spreading long commands across multiple lines. Any character immediately following the backslash will stand without substitution.

Eg:

```
#!/usr/bin/tclsh
puts "He has an \"Apple\" phone."
```

OUTPUT =

```
He has an "Apple" phone.
```

TCL Variable

TCL Command “set”

A variable is an identifier which holds a value. In other words, a variable is a reference to a computer memory, where the value is stored.

Variables are created by "set command" and all variable names are case sensitive. It means hello, Hello, HELLO all are different in TCL.

Eg:

```
#!/usr/bin/tclsh
set age 26
set Age 9570
set AGE 229680
puts "$age Years"
puts "$Age Days"
puts "$AGE Hours"
```

OUTPUT =

```
26 Years
9570 Days
229680 Hours
```

TCL Command “info exists”

The "info exists" command returns 1 if variable name exists as a variable (or an array element) in the current context, otherwise returns 0.

Eg:

```
#!/usr/bin/tclsh
set x 5
puts $x
puts [info exists x]
```

```
unset x
puts [info exists x]
```

OUTPUT =

```
20
1
0
```

Different braces and their behaviour

Curly braces

Curly braces in TCL group words together to become arguments. Curly braces are used to define a block that's deferred - in other words, it may be run AFTER the rest of the command on the current line. Characters within braces are passed to a command exactly as written.

Some points to remember

1. Variable substitution is not allowed inside {} braces
2. It used to create list data type

Eg:

```
#!/usr/bin/tclsh
set x 10
puts {$x}
#Characters within braces are passed to a command exactly as
written.
puts $x

set even {2 4 6 8 10}
#Here even is a list data type
puts $even
```

OUTPUT =

```
$x
10
2 4 6 8 10
```

Square braces

Square braces are used to create nested commands. Simply put, output of one command passed as argument to another command. Square brackets are used to define a block that's run BEFORE the rest of the command on the current line, and the result is substituted into the line.

Eg:

```
#!/usr/bin/tclsh
set x 10
puts "y : [set y [set x 20]]"
puts "x : $x"
```

OUTPUT =

```
y : 20
x : 20
```

Round braces

This command is used to create array data type and also indicate operator precedence.

Eg:

#!/usr/bin/tclsh

set a(1) 10
set a(2) 20

puts [array get a]
puts [array names a]

puts $a(1)
puts $a(2)

OUTPUT =

```
1 10 2 20
1 2
10
20
```

TCL command line arguments

Items of data passed to a script from the command line are known as arguments. The number of command line arguments to a Tcl script is passed as the global variable argc . The name of a Tcl script is passed to the script as the global variable argv0 , and the rest of the command line arguments are passed as a list in argv.

TCL has 3 pre-defined variables such as

1. Argc - indicates the number of arguments passed to the script
2. Argv - indicates list of arguments
3. Argv0 - indicates the name of script

Eg:

```
#!/usr/bin/tclsh
puts "Number of arguments passed to the scripts : $argc"
puts "List of arguments are passed to the script: $argv"
puts "The name of scripts is: $argv0"
```

OUTPUT =

```
$ ./test.tcl 5 10 15 20 25
Number of arguments passed to the scripts : 5
List of arguments are passed to the script: 5 10 15 20 25
The name of scripts is: ./test.tcl
```

TCL Expression and Operators

Arithmetic Operator

A TCL expression consists of a combination of operands, operators, and parentheses.

Eg:

```
#!/usr/bin/tclsh
set x 5
set y 2
puts [expr $x + $y]
puts [expr $x - $y]
puts [expr $x * $y]
puts [expr $x / $y]
puts [expr $x % $y]
```

OUTPUT =

```
7
3
10
2
1
```

Relational Operator

Checks if the value of left operand is greater than the value of the right operand. If yes, then condition becomes true and return 1 else return 0.
Eg:

```
#!/usr/bin/tclsh
set x 5
set y 2
puts [expr $x > $y]
puts [expr $x < $y]
puts [expr $x >= $y]
puts [expr $x <= $y]
```

```
puts [expr $x == $y]
puts [expr $x != $y]
```

OUTPUT =

```
1
0
1
0
0
1
```

Logical Operator

It involves the logics of AND, OR, NOT, then condition becomes true and return 1 else return 0

Eg:

```
#!/usr/bin/tclsh
set x 5
set y 2
set a [expr $x > $y]
set b [expr $x < $y]
puts $a
puts $b
puts [expr $a && $b]
puts [expr $a || $b]
puts [expr !$a]
puts [expr !$b]
```

OUTPUT =

```
1
0
0
1
0
1
```

Bitwise Operator

TCL provides bitwise AND, OR, XOR operators. They follow bitwise rules.

Eg:

```
#!/usr/bin/tclsh
set a 60
# 60 = 0011 1100
set b 13
# 13 = 0000 1101

set c [expr $a & $b]
# 12 = 0000 1100
puts "Value of c is $c"

set c [expr $a | $b;]
# 61 = 0011 1101
puts "Value of c is $c"

set c [expr $a ^ $b;]
# 49 = 0011 0001
puts "Value of c is $c"
```

OUTPUT =

```
Value of c is 12
Value of c is 61
Value of c is 49
```

Ternary Operator

If condition is true, it returns true and false if not.

Eg:

```
#!/usr/bin/tclsh
```

```
set x 12
set result [expr $x > 10 ? true : false]
puts $result
```

OUTPUT =

true

Shift Operator

Shift operator is denoted by either << left shift operator, or by the >> right shift operator. For << left shift operator, the left operand value is moved left by the number of bits specified by the right operand.

Eg:

```
#!/usr/bin/tclsh
set a 5
#0000 0101
set c [expr $a << 1]
#0000 1010
puts "Value of c is $c"
set c [expr $a >> 1]
#0000 0010
puts "Value of c is $c"
```

OUTPUT =

Value of c is 10
Value of c is 2

String Comparison Operator

String comparison operator compares the value of both operands. If the value of operand are the same, then it will return 1 else return 0.

Eg:

```
#!/usr/bin/tclsh
```

```
set x 10
set y 10
set result [expr $x eq $y]
puts $result
set result [expr $x ne $y]
puts $result
```

OUTPUT =

```
1
0
```

Exponentiation operator

Pow () and ** both are same. It always returns floating value. ** indicates the power to the desired operand.

Eg:

```
#!/usr/bin/tclsh
set x 9
set result [expr $x ** 2]
puts $result
```

OUTPUT =

```
81
```

List Operator

If the required value is found in the defined list, it returns 1 else return 0.

Eg:

```
#!/usr/bin/tclsh
set set1 {1 2 3 4 5 6 7 8 9 10}
if {1 in $set1} {
  puts "Value found"
} else {
```

```
  puts "Value not found"
}
```

OUTPUT =

Value found

TCP flow control and decision making

There are various flow control and decision making commands which are used to alter the flow of a program. Program executions always start from the top of source file to the bottom.

If statement consists of Boolean expression followed by one or more statements.

If ... statement

if expr is evaluated to true, then the body of command is executed.

Eg:

```
#!/usr/bin/tclsh
set age 22
if { $age > 18 } {
   puts "You can vote!"
}
```

OUTPUT =

You can vote!

If ... else statement

An if statement can be followed by an optional else statement, which executes when the Boolean expression is false.

Eg:

```
#!/usr/bin/tclsh
set age 17
if {$age > 18 } {
   puts "You can vote!"
} else {
   puts "Sorry, you still cannot vote."
}
```

OUTPUT =
Sorry, you still cannot vote.

Nested if..else statement

It is always legal in Tcl to nest if-else statements, which means you can use one if or else if statement inside another if or else if statement(s).

Eg:

```
#!/usr/bin/tclsh
set ca 75
set end 65
if { $ca >= 50 } {
  if { $end >= 45 } {
    puts "You passed!"
  }
}
```

OUTPUT =
You passed!

Switch statement

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each switch case.

Eg:

```
#!/usr/bin/tclsh
set grade B;
switch $grade  A { puts "Well done!" }  B { puts "Excellent!" }
C {puts "You passed!"  } F { puts "Better try again"  }  default
{ puts "Invalid grade" }
```

OUTPUT =
Excellent!

Nested switch

It is possible to have a switch as part of the statement sequence of an outer switch. Even if the case constants of the inner and outer switch contain common values, no conflicts will arise.

Eg:

```
#!/usr/bin/tclsh
set a 100
set b 200
switch $a {
   100 {
     puts "The value of a is $a"
     switch $b {
        200 {
           puts "The value of b is $b"
        }
     }
   }
}
```

OUTPUT =

The value of a is 100
The value of b is 200

TCL loop controls

Loop statement allows executing a statement or group of statements multiple times. Tcl provides the following types of looping statements.

While loop

A while loop statement in Tcl language repeatedly executes a target statement as long as a given condition is true.

Eg:

```
#!/usr/bin/tclsh
set a 1
while { $a <= 5 } {
   puts "value of a: $a"
   incr a
}
```

OUTPUT =

value of a: 1
value of a: 2
value of a: 3
value of a: 4
value of a: 5

For loop

A for loop is a repetition control structure that allows you to efficiently write a code that needs to be executed for a specific number of times.

Eg:

```
#!/usr/bin/tclsh
for { set a 1}  {$a <= 5} {incr a} {
   puts "value of a: $a"
}
```

OUTPUT =

```
value of a: 1
value of a: 2
value of a: 3
value of a: 4
value of a: 5
```

-END-