МИЭТ
Национальный исследовательский университет «МИЭТ»
Институт интегральной электроники (группы ЭН-34-35, каф. ПКИМС)

Компьютерные технологии в научных исследованиях

Лабораторная работа №1

Синтаксис языка Tcl

Запуск скрипта

Создание и запуск скрипта:

Вариант 2

```
> touch script.tcl
> subl script.tcl &
```

```
#!/usr/bin/tclsh

puts "Hello from Tcl!"
```

```
> chmod +x ./script.tcl

> ./script.tcl
```

Переменные и вывод на консоль

Запись в общем виде:

```
set varName ?value?
```

Объявление переменных:

```
set var1 7
set var2 "This is a string"
```

Использование переменных:

```
set var3 $var2
# var3 = "This is a string"
```

Вывод строки на экран (в общем виде) :

```
puts ?-nonewline? ?channelid? string
```

Вывод строк на экран (с переводом строк):

```
puts "This is a first string"
puts "This is a second string"
```

Вывод строк на экран (без перевода строк):

```
puts –nonewline "This is a first string"
puts "This is a second string"
```

Вывод с явным указанием канала:

```
puts stdout "This is a string"
```

Группировка данных

Вариант вывода:

```
set var 7

puts stdout $var

puts stdout Variable

puts stdout Variable $var

puts stdout "Variable : $var"

puts stdout {Variable : $var}
```

Чтение данных с консоли

Ввод строки с клавиатуры (в общем виде) :

```
gets channelid ?variable?
```

Считывание с указанием переменной:

```
gets stdin var
puts $var
```

Считывание без указания переменной:

```
set var [gets stdin]
puts $var
```

Пример:

```
set x 1

while {$x < 5} {
    puts "x is $x"
    set x [expr {$x + 1}]
}
```

Напишите программу, которая будет считывать с клавиатуры два целых числа и выводить на экран их сумму.

Корректность вводимых данных гарантируется.

Команда сравнения if-elseif-else

Синтаксис:

```
if expr1 ?then? body1 elseif expr2 ?then? body2 elseif ... ?else? ?bodyN?
```

Пример кода:

```
set x 1

if {$x == 2} {puts "$x равно 2"} else {puts "$x не равно 2"}


if {$x != 1} {
        puts {$x != 1 (не равно)}
} else {
        puts {$x равно 1}
}
```

Поменяйте программу таким образом, чтобы я вводил число, операцию, число.

Операции: '+', '-'

Корректность вводимых данных гарантируется.

Команда множественного выбора switch

Синтаксис:

```
switch ?options? string pattern body ?pattern body ...?
```

Пример кода:

```
set x 4

switch $x {
        1 -
        3 -
        5 -
        7 -
        9 { puts "Value is odd" }
        2 -
        4 -
        6 -
        8 { puts "Value is even" }
        default { puts "Value is greater then 10" }
}
```

Поменяйте программу таким образом, чтобы поддерживались операции '+', '-', '*', '/'

Если не одна из них - ругаемся

Корректность вводимых данных гарантируется.

Обработка строк в Tcl (1)

Синтаксис:

```
string compare ?-nocase? ?-length <число>? string1 string2
string equal   ?-nocase? string1 string 2
```

Пример кода:

```
set x "Hello"

if { [string equal $x "Hello"] == 1 } {
  puts "$x is Hello"
} else {
  puts "$x is not Hello"
}
```

Пример кода:

```
set x "Hello"

if { [string compare $x "Hello"] == 0 } {
  puts "$x is Hello"
} else {
  puts "$x is not Hello"
}
```

Обработка строк в Tcl (2)

Получение символа в строке по индексу – команда string index
Синтаксис:

```
string index string_val index_val
```

Пример кода:

```
set x "This is a string"
```

Выведите на экран 4 символ строки (буква 's')

Пример кода:

```
string index "This is a string" end

string index "This is a string" end-3
```

Пример кода:

```
string index "string" 1

string index {string} 1

string index string 1
```

Запросите у пользователя имя файла и проверьте, является ли оно файлом Verilog HDL

Обработка строк в Tcl (3)

Получение подстроки по диапазону – string range
Синтаксис:

```
string range string_val index1 index2
```

Получение длины строки – string length
Синтаксис:

```
string length string
```

Поменяйте задание так, чтобы расширение вместе с точкой добывалось командой string range

Списки в Tcl: объявление (1)

Объявление списков (аналог массива):

Элементы списка - в фигурных скобках

```
set var4 {1 2 3 4 5}

set item_2 [lindex $var4 2]    <- item_2=3
```

Работа с элементами списка – только через команды

Способы объявления списка:

1. **объявлением переменной с указанием элементов**;
2. командой создания списка с указанием отдельных элементов;
3. командой разделения строки на элементы

Списки в Tcl: объявление (2)

Объявление списков (аналог массива):

Список создаётся командой list

```
set var4 [list 1 2 3 4 5]

set item_2 [lindex $var4 2]    <- item_2=3
```

Работа с элементами списка – только через команды

Способы объявления списка:

1. объявлением переменной;
2. **командами создания списка из отдельных элементов**;
3. командой разделения строки на элементы

Списки в Tcl: объявление (3)

Объявление списков (аналог массива):

Список создаётся командой split

```
set var4 [split "1,2,3,4,5" ","]

set item_2 [lindex $var4 2]    <- item_2=3
```

Работа с элементами списка – только через команды

Способы объявления списка:

1. объявлением переменной;
2. командами создания списка из отдельных элементов;
3. **командой разделения строки на элементы**

Обход элементов списка: команды циклов foreach и while (1)

Синтаксис команды foreach:

```
foreach varName list body
```

```
set var { 1 2 3 4 5 6 }

foreach i $var {
  puts "Item = $i"
}
```

Синтаксис команды while:

```
while test body
```

```
set var { 1 2 3 4 5 6 }
set x 0

while {$x < [llength $var]} } {
    puts "item = [lindex $var $x]"
    incr x
}
```

Исполнение внешнего кода и работа с объектами файловой системы

`exec` – исполнение команд ОС, запуск программ и т.д.

`eval` – исполнение строки как кода Tcl.

`source` – загрузка файла и исполнение его, как Tcl скрипта.

```
#!/usr/bin/tclsh

set var [exec ls -al]
puts $var
```

Напишите скрипт, который будет выводить для каждого объекта текущего каталога информацию: это файл или каталог

Проверка существования файла
Синтаксис:

```
file exists name
```

Проверка того, является ли файл файлом, каталогом, является ли файл исполняемым
Синтаксис:

```
file isfile name
file isdirectory name
file executable name
```

Извлечение расширения файла
Синтаксис:

```
file extension name
```

Продвинутая работа с консолью

Управляющие последовательности для манипуляции с цветом:

Цвет текста

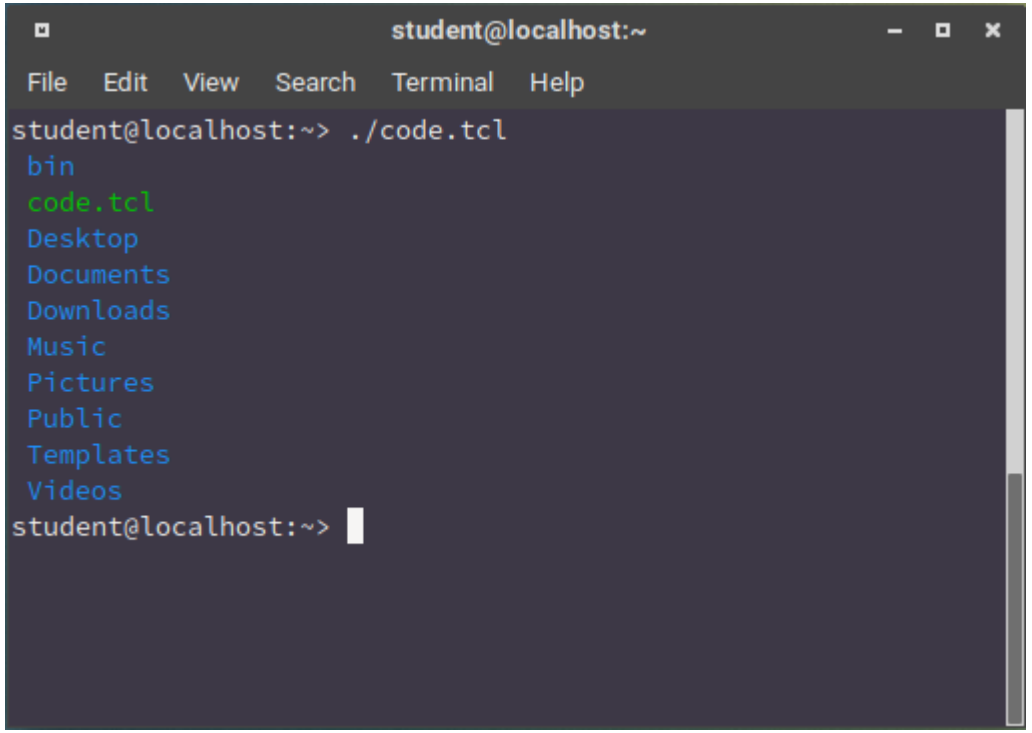
```
\033[30m – чёрный
\033[31m – красный
\033[32m - зелёный
\033[33m - жёлтый
\033[34m - синий
\033[35m - фиолетовый
\033[36m - голубой
\033[37m - серый
```

Цвет фона

```
\033[40m – чёрный
\033[41m – красный
\033[42m - зелёный
\033[43m - жёлтый
\033[44m - синий
\033[45m - фиолетовый
\033[46m - голубой
\033[47m - серый
```

Сброс до значений по умолчанию

```
\033[0m
```



Работа с аргументами командной строки

За аргументы командной строки в Tcl отвечают три переменные:

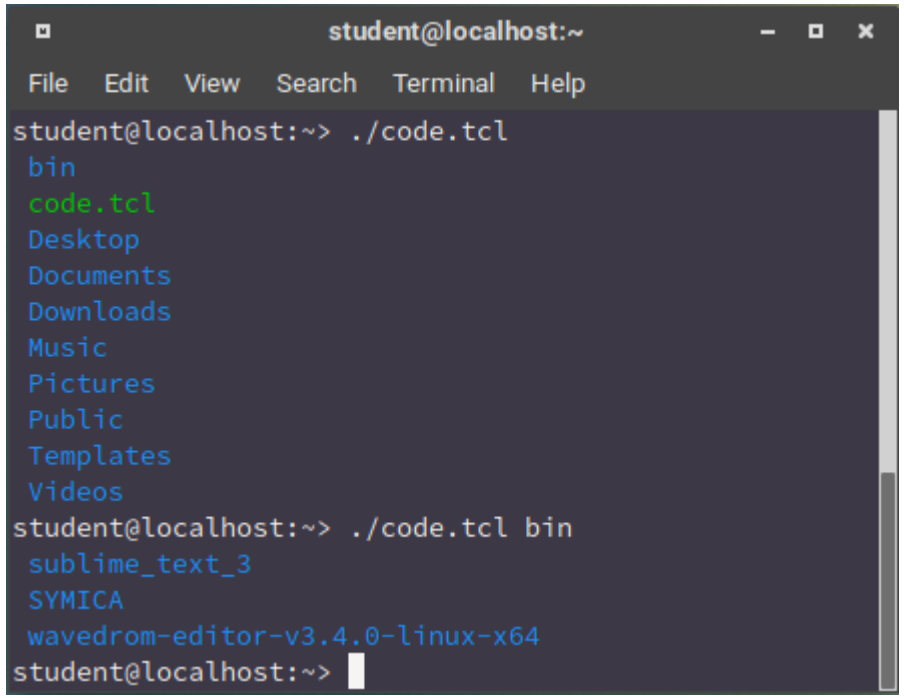
1. argc – число передаваемых аргументов;
2. argv0 – имя скрипта;
3. argv – список аргументов.

```
#!/usr/bin/tclsh

puts "Script name         : $argv0"
puts "Args count          : $argc"

set x 0

while {$x < $argc} {
  puts "Arg [expr {$x + 1}]  : [lindex $argv $x]"
  incr x
}
```



Файловый ввод-вывод в Tcl: чтение файлов

Дескриптор файла

```
set fp [open "data.txt" r]

set file_data [read $fp]

close $fp


set data [split $file_data "\n"]

foreach line $data {

  puts stdout $line

}
```

Все данные файла

Разбиваем файл по строкам

Выводим каждую строку

Считайте и выведите на экран содержимое файла, имя которого задаётся в командной строке в качестве аргумента скрипту

Использование подпрограмм

```
#!/usr/bin/tclsh

proc say_hello {} {
  puts stdout "Hello from Tcl proc!"
}

say_hello
```

```
#!/usr/bin/tclsh

proc say_hello {name} {
  puts stdout "Hello, $name, form Tcl proc!"
}

say_hello Dima
```

```
#!/usr/bin/tclsh

proc say_hello {} {
  set x 4
  puts stdout "Returning 4 from proc!"
  return $x
}

set y [say_hello]
puts stdout $y
```