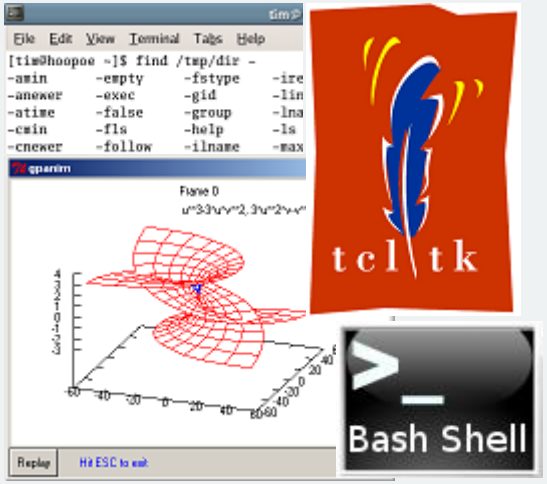
МИЭТ
Национальный исследовательский университет «МИЭТ»
Кафедра ПКИМС

# Интерпретируемые языки программирования




Лабораторная работа №3

**Высокоуровневые скриптовые языки.**
**Язык Tcl.**

# Работа с простыми переменными

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”
```

# Работа с составными переменными

Объявление списков (аналог массива):

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

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

Объявление массивов (ассоциативных массивов):

```
array set ints {
        1 2
        2 9
        3 100
        4 50
}

set item_3 $ints(3)            <- item_3=100
```

# Одномерные и многомерные списки в Tcl

Одномерные списки:

```
set list_var1 {1 2 3 4}

set list_var2 {1 2 3 four five 6}
```

Многомерные списки:

```
set list_var3 {{1 2 3} {4 5 6} 7 8}
```

```
set list_var4 {1 2 {3 4} {} 5 6}
```

# Вывод строки на экран

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”
```

# Команда чтения данных

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]
```

# Команда сравнения IF

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}
}
```

# Команда множественного выбора switch (1)

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"}
}
```

# Команда цикла for

Синтаксис:

```
for start test next body
```

Операторы break, continue:

```
for {set x 0} {$x<10} {incr x} {
   if {$x == 5} {
      continue
   }
   puts "x is $x"
}
```

# Команда цикла foreach

Синтаксис команды foreach:

```
foreach varName list body
```

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

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

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

set j 0
foreach i $var {
  puts "Item no.$j = $i"
  incr j
}
```

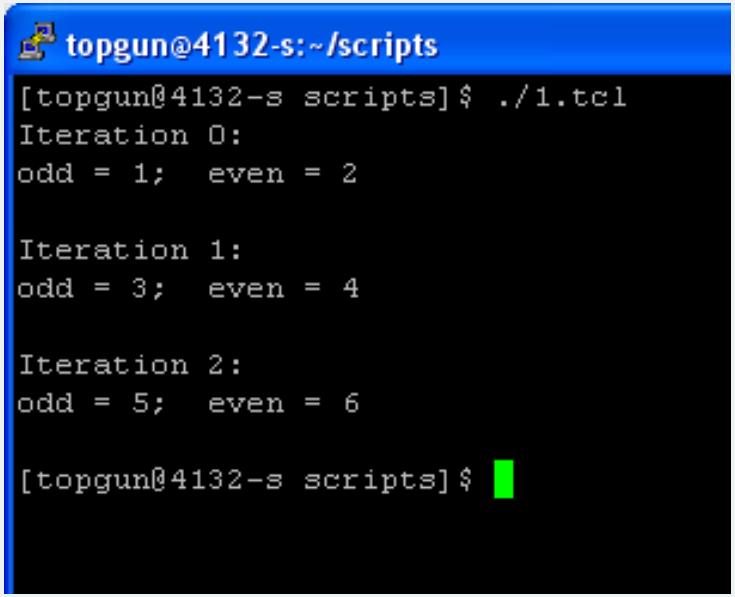
# Команда цикла foreach (2)

Пример работы с двумя индексами:

```
#!/usr/bin/tclsh

set var { 1 2 3 4 5 6 }

set k 0
foreach {i j} $var {
  puts "Iteration $k:"
  puts -nonewline "odd = $i;   "
  puts "even = $j"
  puts ""
  incr k
}
```



# Списки в Tcl: объявление (1)

Объявление списков (аналог массива):

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

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

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):

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

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

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):

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

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

Способы объявления списка:

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

# Команды для работы со списками (1)

Со списками работают следующим перечнем команд:

1. linsert – вставка элементов в список на определённую позицию;
2. lappend – добавление элементов в конец списка;
3. lrange – получает подсписок в заданном диапазоне;
4. lsearch – поиск по списку с заданным критерием;
5. lreplace – замена элементов списка;
6. lsort – сортировка списка.

Информацию о списках или элементах можно получить с помощью команд:

1. lindex – получить значение элемента списка по его позиции (индексу);
2. llength – число элементов списка.

# Команды для работы со списками (2)

Объявление списка:

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

Добавление элемента в список:

```
lappend var4 6                    <- var4={1 2 3 4 5 6}
```

Обращение к элементу списка:

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

Добавление элемента в список:

```
set var4 [linsert $var4 3 9]      <- var4={1 2 3 9 4 5 6}
```

Замена (обновление значения) элемента в списке:

```
set var4 [lreplace $var4 2 2 7] <- var4={1 2 7 9 4 5 6}
```

Определение вхождения элемента в список:

```
lsearch -exact $var4 2            <- 1
```

# Чтение и вывод файла на экран

Дескриптор файла

Все данные файла

Данные файла как список строк

```
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

}
```

# Работа с аргументами командной строки в Tcl

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
}
```

# Получение информации о файлах

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
```

# Регулярные выражения

Регулярные выражения позволяют производить поиск подстрок в строке не на основании точного соответствия, а на основе некоторого шаблона.

```
regexp ?switches? template string ?matchVar? ?subMatcVar1? ...
```

```
regexp {[Y|y][E|e][S|s]} “Yes”
```

Шаблон

Строка, в которой ищется соответствие

Возвращаемое значение: 0 - если нет соответствия
1 – если соответствие есть

```
set a [regexp {[Y|y][E|e][S|s]} “String has yes in the middle”]
```

# Что может входить в состав регулярного выражения

В состав регулярного выражения могут входить:

1. литеральные символы,
2. наборы и классы символов,
3. итераторы,
4. операторы выбора,
5. вложенные шаблоны.

# Литеральные символы

Простые символы – точное соответствие шаблону.

```
regexp {ab} “a”          → 0
regexp {ab} “b”          → 0
regexp {ab} “ab”         → 1
regexp {ab} “abba”       → 1
regexp {ab} “12ab34”     → 1
```

Универсальный заменитель – символ «.»

```
regexp {ab} “a”          → 0
regexp {a.} “ar”         → 1
```

# Наборы символов

Конкретный выбор символа:

```
regexp {[Hh]ello} “A hello string”          → 1

regexp {[Hh]ello} “A Hello string”          → 1

regexp {[Hh]ello} “A hellO string”          → 0

regexp {[Hh]ello} “A hell string”           → 0
```

Допустимый диапазон символов:

```
regexp {[a-z]ello} “A mello string”         → 1

regexp {[a-z]ello} “A Hello string”         → 0
```

# Объединение групп символов

Объединение групп символов:

```
regexp {[a-z]ello} “A Hello string”        → 0

regexp {[a-zA-Z]ello} “A Hello string”     → 1

regexp {[a-z0-9]ello} “A 234ellostr”       → 1
```

Исключение отдельных символов или групп символов:

```
regexp {[^a-z]ello} “A Hello string”       → 1

regexp {[^a-z]ello} “A hello string”       → 0

regexp {[^a-z]ello} “A 234ellostr”         → 1
```

# Некоторые классы символов и их сокращения

| Имя класса | Значение | Сокращение |
|---|---|---|
| alnum | Буквы верхнего и нижнего регистра, цифры | \w |
| alpha | Буквы верхнего и нижнего регистра | |
| blank | Пробелы и знаки табуляции | \s |
| digit | Цифры от 0 до 9 | \d |
| lower | Буквы нижнего регистра | |
| print | То же, что и класс alnum | \w |
| punct | Знаки пунктуации | |
| space | Пробел, перевод строки, \n, \t и т.д. | \s |
| upper | Буквы верхнего регистра | |
| xdigit | Шестнадцатеричные цифры 0-9,a-f,A-F | |

# Итераторы

"*" - любое число вхождений предыдущего компонента

"+" – одно или более повторение

"?" – нулевое или единичное повторение

```
regexp ?switches? template string ?matchVar? ?subMatcVar1? ...
```

```
regexp {\w} "This is sample" var      → 1, var = "T"
regexp {\w*} "This is sample" var     → 1, var = "This"
regexp {\w+} "This is sample" var     → 1, var = "This"
```

# Использование якорей

Якорь “^” - признак начала строки

Якорь “$” - признак конца строки

```
regexp {[0-9]+} "123 456 789" v        → 1, v = “123”
regexp {\d+} "123 456 789" v
```

```
regexp {[0-9]+$} "123 456 789" v       → 1, v = “789”
regexp {\d+$} "123 456 789" v
```

# Использование скобок

```
regexp template string ?matchVar? ?subMatcVar1? …
```

```
regexp {\w+(\d)\w+(\d)} "a1b2" v1 v2 v3
```

v1 = “a1b2”

v2 = “1”

v3 = “2”

```
regexp {^module\s(\w+)} "module inverter (in, out);" a b
```

a = “module inverter”

b = “inverter”

# Задание

Написать на языке Tcl скрипт, который для каждого передаваемого в качестве аргументов командной строки файла сделает следующее:

1. проверит, существует ли файл (если нет – ругаемся и пропускаем)
2. является ли файл того типа, который требуется по варианту (если нет – ругаемся и пропускаем)
3. выполнит действия по варианту

**Чётные варианты**

Скрипт должен убедиться, что ему передаются файлы Verilog (.v, .sv)
Вывести «YES», если в файле есть незакомментированные модули и «NO», если нет
Вывести суммарное число и имена модулей
Вывести отдельно число и имена модулей, у которых более 4-х пинов

**Нечётные варианты**

Скрипт должен убедиться, что ему передаются файлы HSPICE (.cir, .sp)
Вывести «YES», если в файле есть незакомментированные подсхемы (subckt) и «NO», если нет
Вывести суммарное число подсхем
Вывести отдельно число и имена подсхему, у которых более 2-х пинов

**ВСЁ – С ИСПОЛЬЗОВАНЕМ СПИСКОВ И РЕГУЛЯРОК**