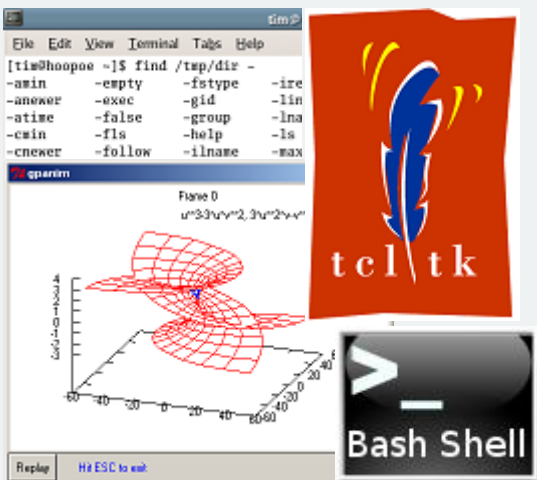




Интерпретируемые языки программирования



Лабораторная работа №3

Высокоуровневые скриптовые языки.
Язык Tcl.



Черты языка Tcl

1. Программа на Tcl состоит из команд, разделённых символами перевода строки или точками с запятой.
2. Ключевых слов как таковых нет — понятие команды в Tcl аналогично понятию процедуры или функции распространённых языков программирования.
3. В перечень команд языка входят аналоги основных операторов языков программирования.
4. Tcl содержит развитые средства работы с регулярными выражениями, ассоциативными массивами.
5. Позволяет описывать процедуры.

```
# Say hello
```

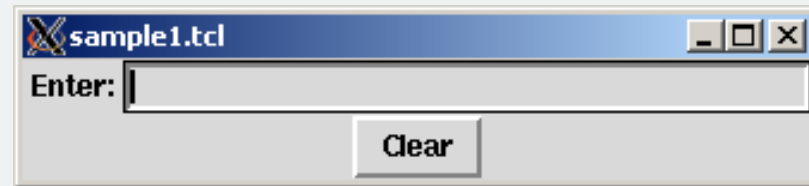
```
set h "Hello"  
set c ,  
set w "world"  
set exm !
```

```
# Print message on screen
```

```
puts stdout "$h$c $w$exm"
```

Фишки языка Tcl (1)

1. Язык TCL единственный интерпретируемый язык, в состав поставки которого входит собственная библиотека для проектирования графического интерфейса пользователя, Tk.





Фишки языка Tcl (2)

2. Для Tcl впервые были разработаны библиотеки для взаимодействия с программами на C++.

Можно из C++ выполнять Tcl-код, можно на C++ писать расширения для Tcl.

```
#include "cpptcl.h"
#include <iostream>

using namespace std;

void hello() {
    cout << "Hello C++/Tcl!" << endl;
}

CPPTCL_MODULE(mymodule, i) {
    i.def("hello", hello);
}
```

```
#!/bin/tclsh

load ./mymodule.so

hello
```



Команды языка

1. простые команды: **set**, **for**, **if**, **switch**, ...

```
set a 10  
set b "bbb"  
set c $a$b
```

2. специальные команды: **#**, ****, **[,],"**, **{, }**, ...

- **\$** - команда подстановки значения переменной вместо её имени;
- **#** - команда комментария;
- **" "** и **{ }** – команды группировки символов с подстановкой значений и без подстановки соответственно;
- **[]** – команда выполняет вычисление аргумента внутри скобок и возвращает результат выполнения.



Работа с простыми переменными

При объявлении переменных не требуется указывать их тип – используется динамическая типизация

Запись в общем виде:

```
set varName ?value?
```



Важно! Всего одно значение!

Для составных данных
обязательно нужно
использовать операторы
группировки.

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 сравнения IF

Пример использования команды
IF в сочетании с ELSEIF и ELSE

```
set x 5
if {$x == 2} {
    puts "$x равно 2"
} elseif {$x == 3} {
    puts "$x равно 3"
} elseif {$x == 4} {
    puts "$x равно 4"
} elseif {$x == 5} {
    puts "$x равно 5"
} else {
    puts "Ни одно из рассмотренных"
}
```



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  
}
```

Чтение и вывод файла на экран

Дескриптор файла

```
set fp [open "data.txt" r]  
set file_data [read $fp]  
close $fp
```

Все данные файла

```
set data [split $file_data "\n"]
```

Данные файла как
список строк

```
foreach line $data {  
    puts stdout $line  
}
```



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? ?subMatchVar1? ...
```

```
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`

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

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

```
regex { [0-9]+$ } "123 456 789" v   → 1, v = "789"  
regex { \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 на лабораторную работу №3

Задание для нечётных вариантов

Разработать скрипт, который зачитывает в качестве входных данных файл на языке Verilog.

С помощью регулярных выражений необходимо определить, есть ли в файле модули (программа должна вывести YES или NO), а затем вывести список имён модулей, которые имеют более трёх пинов.

Задание для чётных вариантов

Разработать скрипт, который зачитывает в качестве входных данных файл на языке Verilog.

С помощью регулярных выражений необходимо определить, есть ли в файле модули, не имеющие пинов (программа должна вывести YES или NO), а затем вывести список имён модулей, которые имеют ровно два пина.

* Не надо мудрить с нетлистами, у нас задача самая простая. Считаю, что:

- список пинов располагается в одной строке с именем модуля,
- направления портов и прочая информация в списке пинов не указывается – только имена.