Unit-5 TCL

Solutions

5.1 Introduction

Q1. What is TCL? Explain the benefits of TCL.

Answer :

TCL

TCL stands for Tool Command Language. It is developed by John K. Ousterhout of the University of California in late 1980s. It was first distributed in 1989.

TCL is a reusable command language. It issues commands along with sufficient programming to define procedures to various interactive tools. These procedures are defined by the users for commands that are very powerful than the commands on the tools base set. TCL also acts as an interpreter. It is embedded in an application to provide a simple command line interface.

TCL has little syntax and no inherent semantics. It is used for developing many applications in National Broadcasting Company (NBC), Cisco etc.

Benefits of TCL

TCL provides several benefits to application developers and users. It is beneficial because of the following reasons,

1. **Rapid Development**

Many applications use TCL scripts as they allow users to program at a much higher level than C, C++ or Java. The script size and development time is reduced to a large extent when TCL scripts are used. The generated scripts are executed directly without recompiling or restarting the application. Thus, this allows users to test new scripts and fix bugs rapidly.

2. **Less Learning and User Convenient**

To use TCL, users have to learn very less and in very hours of learning they can create user interfaces. To develop new applications, the user just needs to learn application specific commands. Thus, this makes it convenient to many users to personalize and enhance their applications.

3. **Interpreted Language**

TCL scripts are interpreted. This feature is very beneficial as it allows TCL users to try new scripts and fix bugs in the scripts easily.

4. **Cross Platform Language**

This feature of TCL is beneficial as it runs an application developed on one platform in another platform.

Example

An application developed in Linux can be run in Macintosh or Windows without making any changes.

5. **Native Unicode Support**

The first dynamic language to support native unicode is TCL. TCL handles text written in any of the world's written language. It requires no extensions to process such text. But, a standard extension, msgcat is used to support simple localization.

6. **Freely Available**

TCL is freely available as open source. It follows BSD license. Thus, any one can view, download, modify and share TCL without any permission.

7. **Glue Language**

TCL is one of the best glue language. Various library packages have been developed by TCL. In addition, users can create their own packages.

An application using TCL includes many other TCL extensions. That provide set of TCL commands. That is, these application can include commands from any of these packages.

8. **Produce a Scripting Language**

TCL helps to have powerful scripting language in applications that require, it to add scripting to an existing application, user just needs to implement new TCL commands that are suitable to the application. These commands are then linked to the TCL library. In this way, it provides both TCL commands and new commands written by user. Thus, providing a powerful scripting language.

Q2. What are the features of TCL?

Answer :

Features of TCL

The features of TCL are as follows:

1. **Operation**

It operates on commands which are written in prefix notation. It is called reusable command language.

2. **Easy to Learn**

TCL has very simple syntax which is easy to learn.

3. **Flexible**

TCL is very flexible and this makes convenient to the users to personalize and enhance their applications.

4. Interpreted Language

TCL scripts are interpreted. They are executed directly without recompiling or restarting the application.

5. Cross Platform Language

TCL supports cross platform i.e., an application developed in one language can run in another language. Example, a Linux application can run in Windows API, Macintosh etc.

6. Datatype

All the datatypes in TCL are treated as strings.

7. Powerful

TCL is a very powerful language. It defines procedures for commands that are very powerful than the commands on the tools base set.

8. Native Unicode Support

TCL supports native unicode. It handles text written in any of the world's written language.

9. Interface Support

TCL includes event driven interface to sockets and files. It also includes time based events and user-defined events.

10. Freely Available

TCL follows BSD license. It is available as an open source. It can be viewed, downloaded, modified and also be shared.

11. Extended and Embedded

TCL can be extended in many ways in C, C++, Java, TCL itself. It can also be embedded directly into the applications.

12. TK Toolkit

TCL is closely integrated with the TK toolkit which has Graphical User Interface (GUI).

5.2 TCL Structure

Q3. Explain the structure of TCL.

Answer :

TCL Structure

The users interact with UNIX shell or DOS using commands. Since, many users are familiar with the command line syntax, TCL implements command line syntax.

A command consists of one or more words separated by white space. If it contains more than one word then, the first word is called command word and the remaining words are called arguments. The command word can be a build-in command or a TCL procedure written by user or an external C function supplied by user. The interpreter makes the arguments available to the command or a procedure or a function by passing control on them. Since, the TCL has no inherent semantics, the semantics is already available in the code that is associated with command word (i.e., first word).

TCL is embedded in host applications to provide a simple command line interface. The interaction between the application and the TCL interpreter is shown in the figure below.

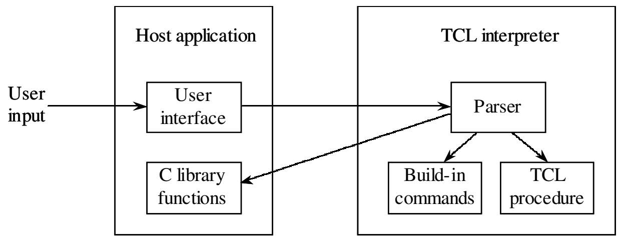

Figure (1): Interaction Between Host Application and TCL Interpreter

The application passes the user input to the TCL parser. The parser then identifies the command word by breaking input into words. Finally, the control is passed to the built-in commands, TCL procedures and C library functions. This is a 'classic' structure of TCL.

Now a days, many applications are build using TCL. In such case, the application shown in figure (1) simply becomes a shell as shown in the figure (2).

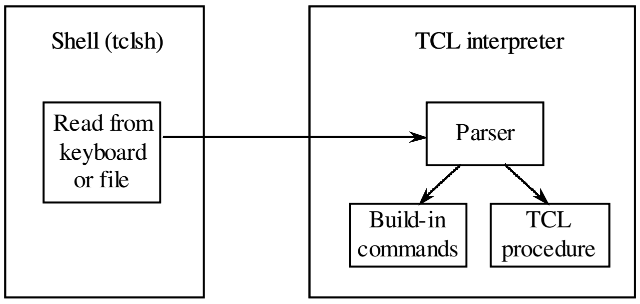

Figure (2): TCL Structure

The input is read from the keyboard or file. The shell then passes the input to the TCL parses. One such shell which is a part of TCL is tclsh.

5.3 Syntax

Q4. Describe the syntax of TCL.

Answer :

TCL Syntax

A TCL command consists of one or more words separated by white space. The first word in the TCL command is called command word and the remaining words are called arguments. The syntax is as follows:

```
cmd arg arg arg
```

A word is a sequence of characters without any spaces or a tab. A word can have a dollar sign ($) or a backslash (\).

Dollar sign substitutes the value of a variable. It is also called substitution operator.

Example: Mr. $name

Here, the $ is used to substitute the value of the variable 'name'.

Backslash is used to introduce special characters.

Example: \n introduces new line

Backslash is also used to disable the special meaning of dollar sign, quotes, square brackets and curly brackets.

Example: Friend\$ is for Friend$

To execute a nested command, we use square brackets. In this case, the output of one command is passed as an argument to another command.

Example: [clock seconds]

This gives the current time in seconds.

To group words separated by spaces together, we can enclose them in double quotes or curly braces. If we enclose them in double quotes, it interrupts the dollar sign and square brackets inside it.

Example: "Good morning $name"

Here, $name is substituted by its value. If we enclose them in curly braces, it interrupt the dollar sign and square brackets inside it.

Example: {Welcome $ure Publications}

Here, $urePublication remains unchanged.

Q5. Define word. Explain three forms of substitutions.

Answer :

Word

A word is a set of characters i.e., a string. It is bounded by start or a white space on the left and a command terminator or a white space on the right considering the following cases,

Case 1

If the word starts with an opening brace ({), then it should be terminated by the matching closing brace (}). The text inbetween these brackets is not processed.

Case 2

If the word starts with an opening square bracket ([), then it should be terminated by the matching square bracket (]). The text in between these square brackets is processed.

Case 3

If the word starts with a double quote, then it should be terminated by the next double quote. One or more substitutions are done to the text between these quotes while the semicolon, backslash, white spaces are treated as ordinary characters.

When the parser scans the word from left to right, there is a possibility of substituting text other than the text enclosed in matching braces. Some of the substitutions provided by TCL are,

(i) Variable substitution

(ii) Command substitution

(iii) Backslash substitution.

(i) Variable Substitution

This type of substitution takes place when a word contains a dollar sign ($). The value of the variable is inserted at the place of the word that is followed by dollar sign. Let us see some of the examples.

Examples

```
1.  set     a   40
    expr    $a*2
```

Output

```
    80
```

In this example, the first command sets the value 40 to the variable *a*. The second command doubles the value of *a*. This is done by variable substitution. The string $a is replaced by 40 and then is multiplied by 2 as 40*2.

```
2.  set  a   90
    set  b   20
    expr $a * $b
```

Output

```
    1800
```

In this example, the first two commands set *a* and *b* values. The third command multiplies both of them by variable substitution.

Example

```
3.  ${variablename}
```

The variable name can be enclosed with { }. In this example, ${variablename} is replaced by value of {variablename}. This notation is generally used with unusual variable name such as *?*.*?*

(ii) Command Substitution

This type of substitution takes place when word is enclosed between square brackets. The TCL interpreter is called recursively to process the text between the square brackets. The text must be a valid TCL script. The square bracket and the text between them is replaced by the result of the script. The backtick and *qx* operators in Perl help in command substitution.

Example

```
set  a   20
set  b   30
set  result [expr $a*$b]
```

Output

```
result = 600
```

In this example, the third command sets the value of result. First the expression inside the square brackets is evaluated and then the result is passed as argument. It becomes as follows,

```
set   results   600
```

(iii) Backslash Substitution

This type of substitution takes place when the word contains backslash (\). It is used to introduce special characters. To use dollar sign in a string it must be appended by a backslash as \$. Similarly, to use backslash in a string it should be written as \\.

The table below lists the backslash substitutions supported by TCL.

Character	Special Meaning
\b	Backspace
\a	Alert
\f	Form feed
\n	Newline
\r	Carriage return
\t	Tab
\v	Vertical tab
\odd	Character with octal code dd
\xhh	Character with hex code hh

Table: Backslash Substitutions

Example

```
set   price\$10
set   Bookdetails Scripting languages\n cost:$price
```

Output

```
Scripting languages
Cost: $10
```

Q6. Explain the process of parsing and executing TCL commands.

Answer :

Evaluating a Command

A TCL command is evaluated in two steps. They are parsing and execution.

Step 1: Parsing

In this step the TCL interpreter divides the command into words based on some rules. The parser then performs substitutions on the words whereever necessary. This step is repeated in the same way for all the commands.

Step 2: Execution

In this step, the first word is treated as command name and the remaining are treated as arguments. TCL then checks if the command is defined and locates the command procedure to carryout its function. TCL interpreter invokes the command procedure if the command is defined and passes all the words to it. The command procedure assigns different meaning to different words i.e., different meaning to different command names and different arguments.

The figure below describes the process of parsing and execution for evaluating a TCL command.

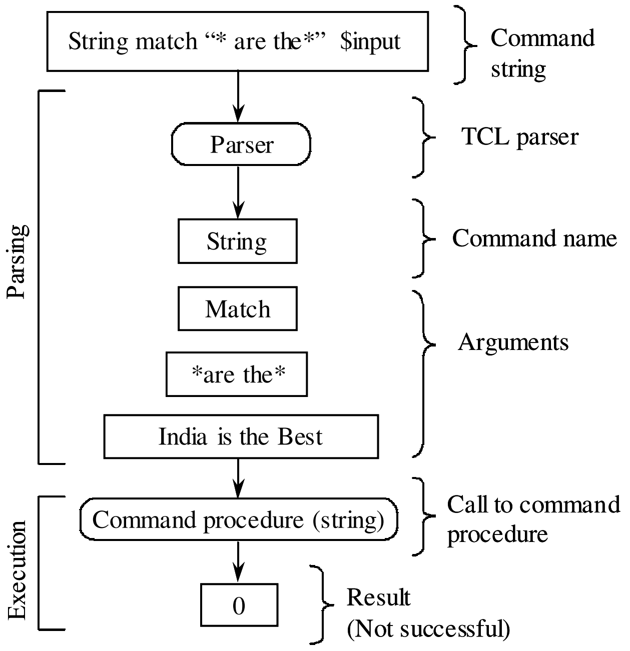

Figure: Evaluating TCL Command by Parsing and Execution

In this figure, the TCL parser divides the given command into words. The parser then substitutes $input with its value. While substituting, the TCL interpreter does not bother about the meaning of the values of the variables that are substituted. It just replaces the variable with its value. It does not bother if its a number or name or anything else.

Let us see few examples where the meaning of the arguments is not bothered by the TCL interpreter.

(i)
```
set   x    40
set   40   x
```

The syntax of 'set' command is set variable name value i.e., first argument is variable name and second argument is value. The above two commands are valid. The first command assigns 40 to x i.e, $x = 40$ while the second command assigns x to 40 i.e., $40 = x$.

(ii) expr 34/4.1

The expr command results in an arithmetic expression after concatenating its arguments. Many other commands take expressions as arguments.

(iii) Lindex{JNTUA JNTUH JNTUK} 1

The syntax of index command is,

lindex list index

The list is {JNTUA JNTUH JNTUK}. It has three elements separated by spaces. The index is 1 which means the element at index 1 has to be returned (i.e., JNTUH has to be returned as index start from 0).

5.4 Variables and Data in TCL

Q7. What is the internal storage of data in TCL? Explain the usage of "set" and "expr" command with suitable examples.

Answer :

Internal Storage of Data in TCL

All types of data such as integers, real numbers, names, lists and TCL scripts are represented internally only in the form of strings in TCL. It has only one data type i.e., string and only one data structure i.e., an array. Strings make the communication simple between scripts and command procedure. Initially, TCL stores only string representations. The received words are interpreted and converted into appropriate TCL variables by the command procedure. The expr command is used for evaluation of expressions. It recognizes the strings and converts them into following appropriate TCL variables,

1. String that represent numbers are converted into numbers.
2. String of decimal digits is treated as decimal integer until it does not start with zero. In this case,

(i) The string is treated as an octal, if the next character is a digit.

(ii) The string is treated as hexadecimal number, if the string starts with 0X.

(iii) All other numbers including decimal point and scientific notation are converted into floating point.

3. A number is required in case of Boolean value such as zero is treated as false and others are treated as true.

Since, all the values appear to be strings, the TCL interpreter manages memory allocation for values and improves its performance by converting between string and internal representations.

set Command

set is a built-in command that is used for assignment.

Example,

```
set a 6;
set b "web programming\n";
```

Only two words are to be given for set command or else it complains if more or less than two words are found. The first character after set represents the variable name and the second character represents the value to be assigned.

expr Command

The expr is a built-in command that is used for evaluating the expression.

Example

```
expr $a *(($C + 4)/2)
```

In the above expression, the expr command first concatenates all the arguments into a single string by performing variable and command substitution.

```
expr{$a *(($C + 4)/2)}
```

After concatenation, the string is evaluated by recognizing brackets and applying operator precedence rules.

5.5 Control Flow

Q8. Explain the "if" and "switch" commands.

Answer :

if Command

The if command evaluates the expression, tests the results. If the result is true, the TCL script is executed. If the result is false, if returns and no further action is taken. Example,

```
if{$x < 0}{
    set x 0
}
```

The above command has two arguments, the first argument is an expression and the second argument is a TCL script. In order to add more scripts to the if command, else if clauses can be included. The final clause will be the else clause with a script that has to be evaluated if no other script succeeds.

```
if{$x < 0}{
---
}elseif{$x = = 0}{
---
} elseif{$x = = 1}{
---
}else{
---
}
```

The above command has four scripts among which only one script will be executed depending on the value of *x*. The resulting value of the evaluated script will be the result of the command. The expressions and scripts are to be enclosed in braces so that they are postponed till the execution of the command.

Switch Command

Like 'if' command, having number of else if clauses, the 'switch' command also has number of patterns. A value is tested against these patterns and the script of the matching pattern is executed. The switch command can be expressed in two forms, but only one of the form is preferred mostly.

```
Switch $a{
        x {incr t1}
        y {incr t2}
        z {incr t3}
}
```

The above given form of a switch command is mostly preferred because the patterns and scripts can be spread across multiple lines. The above command has two arguments, the first argument denotes the value to be tested against the patterns, the second argument is a list of one or more pairs of elements. The first argument of each pair is the pattern that is will be compared against the value. When the pattern matches the corresponding script is executed. The resulting value will be the result of the command. The switch command compares all the patterns sequentially. If no pattern matches, it returns an empty string without any further action. The given switch command increments the variable *t*1 if *a* matches with *x* or increments *t*2 if *a* matches with *y* or increments *t*3 if a matches with *z* and returns an empty string if no pattern matches.

There are three forms of pattern matching that are supported by switch command, exact, - glob and - regexp.

❖ - exact

  - exact is used to select the exact string comparison. It is the default behavior

❖ - glob

  - glob is used to select the pattern that matches with the given string. Each character of the pattern must match with the corresponding character of the string.

❖ - regexp

The - regexp selects the regular expression whose single characters match with the corresponding parts of the string.

Q9. Explain the looping commands with examples.

Answer :

Looping Commands

There are three looping commands provided in TCL, while, for and foreach. These commands run the same script a specified number of times until the condition is true.

(i) while Command

The while command will execute the script until the expression becomes false. It has two arguments, the first argument is the expression and the second argument is a TCL script. The while command evaluates the expression until the result is non-zero. It terminates the command when the expression evaluates to false and returns an empty string.

Example

```
while{ $x > 0}
{
set result[expr $result * $base]
set P[expr $P -1]
}
```

(ii) for Loop

The for loop executes the script for a specified number of times. It provides more explicit loop control. It has four arguments the first argument is an initialization script, the second argument is an expression that determines when to terminate the loop, the third argument is a reinitialization script that is evaluated after the execution of the body of the loop before the evaluation of test and the fourth argument is a script i.e., the body of the loop.

Example

```
for{set i 0} {$ I < i 0} {incr i 3}{
      set x[expr $x + $y(i)]
}
```

(iii) foreach Loop

The foreach command iterates through arrays. For each of the loop the value of the current array element is processed and incremented or decremented by one. It has three arguments, the first argument is the name of the variable, the second argument is a list and the third argument is a TCL script i.e., the body of the loop.

Example

```
foreach{a  b} {1 2 3 4 5}
{
puts "< $a > < $b >"
}
```

This prints

```
1 2
3 4
5
```

Q10. Explain "break" and "continue" commands.

Answer :

The break and continue commands in TCL are used to abort a part or all of looping command.

Break Command

The break command causes the termination of the innermost enclosing loop immediately.

Example

Print a line for each of the integers from 0 to 5,

```
for{set x 0} {$x< 10} {incr x}
{
    if {$ x>5}{
    break
}
Puts "x is $x"
}
```

Continue Command

The continue command causes the termination only of the current inner most loop, it continues with the next iteration. It is used to skip a part of a loop and continue with the next iteration.

Example

Print a line for each of the integers from 0 to 10 except 5.

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

5.6 Data Structures

Q11. Explain in detail about list.

Answer :

List

List is a string containing list of elements separated by spaces or tabs or new line.

Example: India UK US Africa Germany

If we use braces and quotation (double quotes) to group element into sublists.

Example: India {UK $a}. "UK $a"

If the variable *a* contains the value "Africa", then the second argument with braces will be set as it is i.e., UK $a because braces provide strict quotating and the third argument with quotation is the string "UK Africa".

Construction of List

(i) **list Command:** List command groups the given arguments as the values in the list. If any argument of a list is appended with a special character then, the command performs automatic quoting is the argument itself is a list.

In, the above example, variable *X* is appended with a special character $, this will indicate that it is already a list enclosed within braces.

Example: List $ X apple banana grapes

(ii) **linsert Command:** This command return a new list formed by inserting the arguments into an existing list at a particular point.

Syntax

linsert list index value? value?.....

(iii) **Lappend Command:** This command appends the arguments at the end of the list.

Syntax

lappend varName value? value .... ?

(iv) **split Command**

split command returns a list formed by eliminating the splitting characters.

Example

David Alam # Radhi # Raghu # 12346567

If this line is read into the variable line, then

split $line #

This command returns a list of item as,

{David Alam} Radhi Raghu 12346567

David and Alam are enclosed in a single braces, so they form a sublist because they were not separated by split character.

If a single list has to be made eliminating the sublists. We use,

split $line{#}

This command returns,

David Alam Radhi Raghu 12346567 by eliminating sublists and spliting characters.

(v) **llength Command**

llength command will return the number of elements present in the list.

Syntax

llength list

Example

llength $ list

This returns the number of element present in list.

llength{a b c d}

This returns 4.

llength{ }

This returns 0.

(vi) lindex Command

This command returns a value which is identified by specified index value.

Syntax

lindex list? index .... ?

lindex takes two arguments list of element and an index value, this will return the value specified at given index.

Example

lindex{Apple Mango Orange}1

This will return Mango.

(vii) lrange Command

This command returns the elements of a list from the specific range.

Syntax

lrange list start number end number.

lrange takes three argument a list, start index and end index.

Example

lrange list 1 14

This returns all the elements present in between 1 to 14.

Q12. Explain in detail about arrays.

Answer :

Array

An array is a collection of elements, each element has its own name and value.

Initialization

Initialization of array is done in single step using array set command.

Example

array set names{ David, Alam, Rahul, Naveen}

This creates the array to names.

Extracting

In order to retrieve the elements present in the array we use array get command

Example

```
Foreach{key value} [array get A]
{
.....
}
```

This command present prints the keys and value alternatively.

To retrieve the list of keys present in the array we use array names command.

Example

```
foreach key [array names Names]
{
}
```

Modifying the Array

Let us consider an array with list of earnings.

array set earnings{January February March .... December}

To set a value in the array, we use set command

set earnings (January) 7600

This will assign 7600 value in the element January in the array earnings.

set earnings (January)

This displays the value assigned to January.

One Dimensional Arrays

In TCL, one dimensional arrays are represented as,

array set array name [list of elements]

To set a specific values in the array, we write it as,

array set b(bar) 4

This sets bar as the fourth value in the array.

Multidimensional Array

Two dimensional effect is given to the array by using two index separated by a comma.

set$aname (1, 1) 2000

This will set 2000 value in the aname arrary at first rows, first column.

5.7 Input/Output

Q13. Explain TCL I/O commands in detail.

Answer : Model Paper-II, Q5(b)

TCL I/O is built using strings. It communicates with the external resources through streams, like stdin, stdout and stderr.

Output Command

puts

puts is used to preform the basic output operations. By default, this prints the output to stdout by appending a new line.

Example, puts "Hai students"

Option

nonewline

This option omits the newline which is appended by default.

Example, puts - nonewline "enter your student id:"

Input Command

gets and read are used to perform the basic input operations.

gets

The line given in stream will be read by get command and it will return the line by appending the new line

Gets stdin;

In order to assign the input string to a variable we use or giving or specifying a destination to the given input

set a[gets stdin ans]

When the file reaches its end, gets will either return a empty string or – 1 according to the format of the get command.

read

In a stream, if large volume of input data is given, then to have a efficient block reading facility we use read command.

Example, read stdin

This returns a string including the final new line.

Options

nonewline: This omits the final new line.

Syntax

read-nonewline stdin

If read command is following with the byte count argument, it reads the data given in the stream till it reaches the specified byte count value.

Examples, read stdin 4096

This reads 4096 bytes from the stream.

eof Command

If the file reaches its end, then eof command returns 1otherwise 0.

```
while{! [eof stdin]}
{
}
```

5.8 Procedures

Q14. Explain about producers.

Answer :

The TCL command 'proc' is used to create procedures.

Syntax

Proc name{argument list} {body}

Here, 'name' indicates the name of the procedure, 'argument list' contains the names of the arguments (formal parameters), 'body' contains the TCL script.

Example

```
proc demo {x  y}
{expr {$x – $y}}
}
```

The procedure 'demo' can be involved as any other TCL command. The values given in the procedure call are assigned to the variables in the argument list as soon as the procedure is invoked. The argument list may contain a variable name or a list of variables names or a default value to be used, if a value is not provided during a procedure call.

To get the result of a procedure the command 'return' is used, which returns a particular value. In the case, if 'return' command is not used in a procedure then the last statement in the 'body' returns the result.

Invoking Demo

As mentioned earlier, the procedure 'demo' must be invoked only with two arguments, we see some of the invocations,

```
demo 4 2
⇒ 2
demo 4 4
⇒ 0
demo 4
⇒ wrong # args: should be "demo a b"
```

The last invocation 'demo' shows an incorrect invocation.

'proc' is an ordinary TCL command. TCL processes the arguments to 'proc' in the similar way as it processes the other commands. To pass the argument name of 'demo' to 'proc' as single list of argument names, the braces{a b} are used. This does not mean any special syntactic notation. Generally, if a procedure contains only one argument the braces are not required but to maintain consistency most of them uses braces. Similarly to avoid substitutions, the entire script body is passed as a single argument to the 'proc' using the braces around the last argument of proc.

Example

```
proc fact {a}
{
if { $a < 1 }
{
return 1
}
return [expr {$a*[fact [expr{$a – 1}]]}]
}
```

Output

```
fact 3
⇒ 6
fact 5
⇒ 120
fact 0
⇒ 1
```

Here, the factorial function uses 'return' command, which makes the enclosing procedure to return immediately whenever the value of $a is less than or equal to 1. Otherwise, the 'expr' is evaluated which recursively calls 'fact' and finally returns the output. So, whenever we want to avoid executing the entire script, 'return' command is invoked.

Q15. Explain the concept of local and global variables with suitable examples.

Answer :

A TCL procedure uses distinct set of variables when it is evaluated.

Local variables are defined and accessed within the procedure. These variables are taken from the caller of a procedure and are removed implicitly whenever the procedure returns. The name of local variable can match with the name of a global variable or namespace variable or other local variable in some other procedure. However, these variables are distinct.

The command that invokes a procedure is used to set the values of the arguments in that procedure. These arguments are local variables to a procedure. The values of local variables corresponding to the arguments passed to a procedure are used in the execution of a procedure. The other local variables are set automatically. The recursive call to a procedure have distinct set of local variables for each invocation.

Example

```
proc exponent {base a}
        {
        set output 1//local variable
        while {$ a > 0}
        {
        set output [expr $output * $base]
        set a [expr $a – 1]
        }
        return $output
        }
```

Global variables are defined outside a procedure. These variables remain exist, until they are removed explicitly. 'global' is the command used to refer the global variables in a procedure.

```
global a b
```

The global variables 'a' and 'b' are accessible inside a procedure. The command 'global' is invoked during the execution of procedure. The effect of 'global' command starts when the procedure is invoked and remains until procedure returns. The arguments in a 'global' command are treated as the names of global variables and references are made to these names in a procedure. The 'command' global can be invoked at any time, after invoking a procedure.

Example

```
proc exponent{base a}
{
global output       //global variable
while{$a > 0}
{
set output[expr $output * $base]
set a [expr $a – 1]
}
}
```

Q16. Discuss about default and variable numbers of arguments for procedures.

Answer :

Default Arguments for Procedures

In a procedure, all the arguments or some of the arguments may contain default values. The argument list in a procedure may contain a number of sublists in which each sublist is considered as a single argument. The single element in a sublist specifies the name of a argument, if the sublist contains an additional element (two elements) then the second element specifices the value of first element (argument).

Example

```
proc dec {value {decrement 1}}
{
expr $value - $decrement
}
```

The procedure above, decrements a given value by 1. 'value' is the first element which has no default value. 'decrement' is the second element with default value '1'. The procedure 'dec' can be invoked with any of the two arguments.

```
dec 30 2
⇒28
dec 30
⇒29
```

If a value is not specifies as default for an argument in a procedure, then at the time of procedure invocation it is to be supplied. Always the default arguments are provided at the last in a argument list. The arguments followed by default argument must be default arguments. After invoking a procedure of any of the argument specified is removed then all the argument following the removed one must be removed.

Variable Number of Argument for Procedures

In a procedure, the argument list may contain any number of arguments. If 'args' is a special name of the last arguments in the argument list, then different number of arguments may be used to invoke a procedure. Except the last argument 'args' remaining arguments in the argument list are operated in the same way as earlier, we can also specify new arguments. 'args' is a list containing set of additional arguments, if there are no extra arguments then it is set to an empty string (default value).

```
proc demo{args}
{
set sum 0
foreach value $args
{
set sum[expr {$sum + $value}]
}
return sum
}
```

Output

```
demo 3 2 3
⇒11
demo
⇒0
```

The additional arguments before 'args' in the argument list of procedure are treated as default arguments.

Q17. Explain call-by-value mechanism with an example.

Answer :

Call-by-Value Mechanism

In TCL, reusability of modules is achieved by procedures. They can be defined at any time. The arguments are passed to the procedure either by value or by reference.

The syntax of procedure is as follows,

```
proc proc_name arguments ?arguments? body
```

The arguments used are generally local. Global variables can also be referred using the global command.

Example 1

```
proc inc{i {j 1}}
{
    expr $i + $j
}
```

Here inc is the procedure name. The arguments are passed by value. The arguments are specified with the default values,

(a) If we call inc 92 2,

The result is 94

(b) If we call inc 5,

The result is 6

Example 2

```
proc add list
{
    set i 0
    foreach j $list
    {
    inc i $j
    }
}
```

Here, the procedure add has a list, whose elements are treated as arguments. Suppose, if there are no elements, it will be treated as empty string.

(a) If we call add 2 4 6

The result is 12

(b) If we call add

The result is 0.

Q18. Does TCL support call-by-reference? Explain.

Answer :

Usually, TCL does not provide the mechanism of call-by-reference. This seems difficult to change the value of a current variable by writing a procedure. Any way, in TCL the variable name is a string value which can store other variable and hence by using substitutions one can copy the mechanism of a reference.

Example

```
set a "Helloworld"
set b a
set $b
⇒ Helloword
```

Here, the string value '*x*' is substituted in $y by the TCL interpreter. The value of the variable is returned by the 'set' command which takes the variable name as the argument and executes.

The TCL command 'upvar'in conjunction with the above concept supports the mechanism of call-by-reference. Variables such as global, local and namespace can be accessed from outride the context of a procedure (i.e., from other active procedure).

This is done by using 'upvar', often 'upvar' is used to provide call-by-reference. This concept is specifically useful when dealing arrays.

For example, if 'demo' is a procedure to which we have to pass an array, say '*n*'. This cannot be done by simply writing the command demo $n. Since, there are values for individual elements and not a single value of entire array. For this purpose the name of an array is send to a procedure as demo *n*. However, 'upvar' is used to access the array elements.

Example

```
proc demo{ename}
{
upvar $enamen
foreach el[lsort [array names n ]]
{
puts "$el = $n ($el)"
}
}
set emp(Num) 25
set emp(Work) "Content writer"
demo emp
⇒ Num = 25
⇒ Work = Content writer
```

Here, the name of an array is passed to the procedure 'demo' when invoked. Using a local variable '*n*', this array is accessed. This is done by 'upvar' command, it takes the name of a variable as the first argument which is accessible to the caller of the procedure. In the calling procedure this variable may be a global, local or namespace variable.

'upvar' command takes the name of a local variable as second argument. The variable 'ename' on the caller is referred by this local variable '*n*', where it can be accessed using 'upvar'. The elements of global variable 'emp'are the actual elements which the procedure 'demo' reads through array '*n*'. 'Emp' is altered when procedure demo writes to '*n*'. The list of elements in a array are returned using the command 'arraynames' by procedure 'demo'. Finally, 'lsort' is executed which sorts the elements in an order and print them. By default, the first variable name in an 'upvar' command refers the caller of existing procedures content. The variables from any level including global level on the call stack can be accessed.

Example

```
upvar #0 order a
```

To consider the variable 'order' as a global variable notation #0 is used. The above command can access global variable 'order' through a local variable 'a'.

```
upvar 2 other a
```

This command accesses the global variable of the caller the existing procedure making it as local variable '*a*'. The current content is denoted by level '0'. '2' in the above command shows the content of 'other' variable which is two levels above the call stack.

Q19. Write short notes on scope of variables.

Answer :

Scope of the Variable

TCL supports different variables and evaluates them within its scope.

❖ Local variables are the variables which are defined inside the TCL procedure or are the arguments of procedure itself. The scope of these variables is within the procedure and these variables are deleted when the procedure returns. Thus, it is possible to have same local variable for two different procedures.

❖ Global variables are the variables which are defined outside the TCL procedure. The scope of these variable is throughout the script. These variable must be explicitly deleted.

❖ Namespace variables are the variables whose scope is throughout the context of the particular namespace.

Local variables can have same name as global variables or namespace variables. They are treated as two different variables and changes made in one variable do not affect the other.

The global and upvar commands change the scope of the variables.

1. Global command cause the local variables to refer the global variable i.e., the global variables can be used by the procedure using global command.

   Example: global x y

   Here, the global variables x and y can be used inside the procedure.

2. The upvar command works similar to global command. It helps in changing the name of the variable in current scope to the variable in different scope.

   Example

```
proc increase {var{dec 1}}
{
upvar $var g
    set of [expr $V + $dec]
}
```

5.9 Strings

Q20. Explain string operations with suitable examples.

Answer :

String Operations

In TCL, string operations are carried out by using the string command. The various string operations are discussed below,

1. String Length stc

This command returns the length of the given string, str. That is, it returns the count of number of characters.

For example,

(i) String length Sure

It returns 4

(ii) String length "Sure Publications"

It returns 17.

2. String Compare ?-nocase? ?-length num? str1 str2

This command returns -1 if str1 < str2

0 if str1 = str2

1 if str 1 > str2

The -nocase option indicates case insensitivity.

The -length option indicates the first num characters to be compared.

For example

(i) String compare Nehru Gandhi

It returns –1

(ii) String compare -length 3 Master Mask

It returns 0

3. String Match ?-nocase? Pattern str

This command uses glob-style matching rules. It returns 1 if pattern matches the given string str and it returns 0 if it fails to match. The -nocase option is optional which indicates case insensitivity.

For example,

(i) String match S* Sure

It returns 1

(ii) String match c.t coat

It returns 0

4. String To lower str ?start? ?end?

This command converts all the upper case characters in the given string str to lower case characters and returns the converted string. The start and end options are the indexes which when specified, the conversion starts from start index and ends at end index.

For example,

(i) String to lower "Sure Publications"

It returns sure publications.

(ii) String to lower "SURE PUBLICATIONS" 5 10

It returns SURE publicATIONS

5. String To upper str ?start? ?end?

This command is similar to to lower except that it converts all the lower case characters in the given string str to upper case characters and returns the converted string.

For example,

(i) String to upper "Sure Publications"

It returns SURE PUBLICATIONS

(ii) String to upper "Sure" 2 3

It returns SURe.

6. (a) String Trim str ?char?

This command removes the characters other than the characters that appear in chars from the given string str and returns that string.

For example,

String trim education aeion

It trims the vowels from the given string (education) and returns dctn.

(b) String trimleft str ?char?

This command works similar to string trim except that it removes the leading characters.

(c) string trimright str ?char?

This command works similar to string trim except that it removes the trailing characters.

7. Append Vriablename strl ?str2? ?str3?

This command concaten sates the given string or strings onto the end of the value of the variable.

Note

The first argument must always be a variable.

For example,

set greeting {Good Morning}

append $greeting "! Have a pleasant day"

It returns Good Morning! Have a pleasant day.

8. Format formatstring ?value value...?

This command substitutes the value arguments in place of %sequence in the given format string. The resultant string will be similar to the specified formatstring.

For example,

(i) set name book

set cost 25

set display [Format "This %S costs %dRs"

$name $cost]

puts display

Its format formats the given string in a way similar to printf function in C. It returns the following formatted string which is stored in display.

This book costs 25 Rs.

(ii) Format "Square root of 3 is %.5f" [expr sqrt(3)]

It returns square root of 3 is 1.73205.

9. Split String Splitcharacter ?split characters?

This command splits the given string into substrings. It creates a list for storing the substrings. The split character specified is used to separate the given string on its occurrence.

For example,

1. Set msg "Apple, Mango, Grapes"

split $msg,

The split command splits the given string at the 'g' and stores it in the list. The list contains 3 elements, Apple Mango Grapes.

2. Split abxycydxy

Here, more than one splitcharacter is specified. It returns ab{ } c d

5.10 Patterns

Q21. Explain pattern matching techniques with examples.

Answer :

Pattern Matching Techniques

TCL provides two pattern matching techniques. They are glob matching and regular expression matching.

1. Pattern Matching using Glob Style

This technique is the simplest one. It is named after the globbing mechanism used in unix shell for expanding file names. The glob-style matching is easy to learn and use. It is suitable only for simple pattern matching which is accomplished by string match.

The syntax of string match is,

string match pattern string or variable.

It returns on success and 0 on failure. It performs glob matching using only three patterns that are listed in the table below.

S.No	Pattern	Description
1	*	Matches 0 or more characters
2.	?	Matches a single character
3.	[characters]	Matches a single character from the given set of characters enclosed in square brackets.

Table (1): Pattern used for Glob Matching

Let us see few examples.

(i) Set Name Education

string match *cat* $name

⇒ 1

Here, the pattern *cat* matches the string which has a substring cat. It returns 1 as the match is found in the value of the variable name.

(ii) String match Educat? $name

⇒ 0

Here, the pattern Educat? match the string of length 7 and which starts with Educat. It returns 0 as the match is not found in the value of the variable name.

(iii) String match ?ure sure

⇒ 1

Here the pattern ?ure matches a string of length 4 that ends with ure. It returns 1 as the match is found in sure.

(iv) String match *sure sure

⇒ 1

Here, the pattern *sure matches a string that ends with sure.

(v) String match {[A – Z]}*} university

⇒ 0

Here, the pattern {[A – Z]*} matches a string starting with a capital letter.

(vi) String match –nocase U* university

⇒ 1

Here, the –nocase option indicates case insensitive. The pattern U* matches a string starting with u or U.

(vii) String match {*\?} why?

⇒ 1

Here, the pattern {*\?} matches a string ending with? (Question mark).

(viii) Set files {prog.c xyz.html progl.c stdio.h}
string match *.[ch] $files

⇒ 1

Here, the pattern *.[ch] matches all strings in $files ending with either .c or .h.

2. Pattern Matching using Regular Expression

This technique is very powerful and is used for complex pattern matching. The regular expression matching is done using the regexp command.

The syntax of regexp command is,

(a) regexp pattern string
It returns 1 on success and 0 on failure

(b) regexp pattern string variable.

In this case, apart from returning the value 1 or 0, the string that matches the pattern is stored in the variable.

It performs regular expression matching using the patterns that are listed in the table below.

S.No	Pattern	Description
1.	.	Matches a single character
2.	^	Matches the position at the beginning of input string
3.	$	Matches the position at the end of input string
4.	*	Matches 0 or more characters of the previous pattern
5.	+	matches 1 or more characters of the previous pattern
6.	?	Matches 0 or 1 character of the previous pattern
7.	[ ]	Matches a single character from the set of characters enclosed in square brackets [ ].
8.	( )	Matches a string which has a substring similar to the string enclosed in brakets ( ).
9.	|	Matches the altenate character
10.	alpha	Matches a letter
11.	upper	Matches an upper-case letter
12.	lower	Matches a lower-case letter
13.	digit	Matches a digit
14.	xdigit	matches a decimal digit
15.	alnum or print	Matches an alphanumeric
16.	blank	Matches a space or tab character
17.	punct	Matches a puntuation.

Table (2): Patterns used for Regular Expression Matching

The patter numbered 1 to 9 are supported in TCL 8.0 where as the remaining patterns 10 to 15 are supported in TCL 8.1

Let, us see few examples,

1. Set message {Hi! your id is 591}

   regexp {^i} $message

   It matches strings that start with i. It returns 1 as the strings id and is start with i.

2. regexp {^ [A – Z a –] + $} $message

   It matches the string that starts and ends with a characters. It returns 1 as the strings your 'id' is start and end with a character.

3. regexp {^[[: digit:]] + $} $message

   It matches the string that starts and ends wih a digit. It follows the syntax of TCL 8.1. It returns 1 as the string 591 starts and ends with a digit.

4. regexp {C.t} "Education} var

   When any variable (Ex: var as mentioned above) is specified, it matches the given expression to the string and stores the part of the string that is matched in that variable. To view the contents of var, post is used.

   post $var #returns cat

Q22. Explain the use of switch command for pattern matching.

Answer :

Switch Command

The switch command executes one of the several TCL scripts, depending on which the pattern matches.

The syntax of switch is as follows:

1. Switch ?option? string pattern body ?pattern body...?
2. Switch ?option? string {pattern body ?pattern body...?}

It matches the given string against the given pattern. Once the pattern matches the string, it evaluates the body by recursively passing it to the TCL interpreter. The last pattern argument is generally defaults. It matches anything. If the pattern argument does not match the string and if default is not mentioned then this command returns an empty string.

The option argument can be either of the following. They start with a hyphen (-).

(a) -exact

This is a default opiton. It follows exact matching to match the string against the pattern.

(b) -glob

This option follows glob-style matching to match string against the patterns.

(c) -regexp

This option follows regular -expression matching to match the string against the patterns.

(d) -nocase

This option follows case sensitivity while matching the string against the patterns.

(e) -matchvar varname

This option is used along with -regexp option. The list of matches found by the regular expression engine are stored in the variable specified (i.e., varname).

(f) -indexvar varname

This option is also used along with -regexp option. The list of indices that refer to the matching substrings found by the regular expression engine are stored in the variable specified (i.e, varname).

(g) --

This option marks the end of options. An argument which follows this option will be treated as stirng. It is not used when pattern and body arguments are grouped together into single argument.

Let us look at few examples.

1.
```
set var "xyz"
switch xyz
{
x-
b {expr{1}}
$var {expr{2}}
default {expr{3}}
}
```
   Here, the string argument, xyz is matched with the variable, var. It returns 2 after evaluation the body.

2.
```
switch -glob xxxy
{
        x*y       {expr{1}}
        y            -
        x*        {expr{2}}
        default   {expr{3}}
}
```
   Here, the switch command uses the glob-style matching. It returns 1 after evaluating the body.

3. switch abc

```
{
        m         -
        n         {expr{1}}
        default   {expr{2}}
}
```

Here, the string argument, abc fails to match the patterns specified. Thus, the default clause is taken and it returns 2.

5.11 Files

Q23. Explain about associating a stream with a file or pipe.

Answer :

Associating a Stream with a File or Pipe

The open function is used for associating a stream with a file or pipe. The name assigned to the stream is called the file descripter. The predefined stream for input output are stdin, stdsut and stderr.

The syntax for open command is as follows,

(a) Open filepath ?mode? ?permission?

The filepath specifies the path of the file to be opened. The mode determines how the file can be accessed. This is a optional argument.

It can be r, r+, w, w+, a, a+.

By default, it is read only i.e., r

The permission detemines who can access the file. This is also an optional argument. By default, it is 0666 i.e.,

Example

```
Set cfile [open/temp/prog.c w 0600]
puts $cfile "It opens a cfile".
```

(b) Open/command ?access?

The command is treated as a list and is executed by creating subprocess. Based on the accessed specified it exeuctes pipes for writing/reading the input/output to/from a pipeline. It return an identifier for communicating with the subprocess.

Example

```
Set pipeto [open/abc w]
```

This opens a stream for writing. The input that is sent by puts to this stream will be piped into the standard input of command abc. The standard output of abc is sent to the standard output of TCL application. The syntax is similar to perl. But, in perl the bar is put after the process name to indicate read from a pipe.

In TCL the bar is in the front to indicate 'pipe' and that the stream is open for reading as shown below.

```
Set pipefrom [open/abc r]
```

It attaches the stream to the standard output of abc.

Suppose a 'pipe' stream is opened in r+ mode i.e., for reading and writing as shown below.

```
Set pipe [open/abc r+]
```

It sets up two pipes, one for writing to abc's standard input using puts $pipe.

Another for reading from abc's standard output using gets $pipe.

Q24. What is the purpose of the fconfigure command?

Answer :

fconfigure Command

The fconfigure command is used for performing set and get options on a channels.

The syntax of fconfigure is as follows:

(a) fconfigure Chid

The chid parameter specifies the channel for which an option has to be set. It returns a list that contains alternate option names and values for the channel.

(b) fconfigure Chid Chname

Here, the parameter chname is specified. It returns the current value of the given option.

(c) Fconfigure Chid Chname Chvalue ?chname chvalue?

Here, a pair or pairs of name and value are supplied. It sets each of the named option to the corresponding value option and returns an empty string.

The options supported by all the channels are discussed below.

1. Blocking Boolean

This option tells whether the I/O operations (gets, read, puts, flush, close) on the channel block the process indefinitely. The value option must be only a boolean.

2. Buffering Value

(a) If value = full, the output is buffered by the I/O system until internal buffer is full or until flush command is innvoked. By default, -buffering is set to full exculding the channels that connect to terminal like devices.

(b) If value = line, the output is automatically flushed by the I/O system when output is newline. This value is set as initial value for channels that connect to terminal like devices.

(c) If value = name, for every output operation I/O system will flush automatically.

3. -Bufferize Size

The value option, size must be an integer. This option sets the size of buffers that are allocated for channel to store in bytes.

4. -eofchar char

-eofchar {inchar outchar}

This option is used to set the end of file marker. It is supported by DOS files systems. The value option, char can be a empty string or a non-empty string. By default, the value is empty string except for the files under windows. The inchar and outchar specifies the end of file marker for input and output respectively.

5. -translation mode

-translation {inMode outMode}

This option is used for marking end of line. The end of line in TCL scripts is represented using newline character (\n). In different plat forms or in same platform different devices, the representation of end of line is different. The value option mode can be auto, binary, cr, crlf and lf. It is by default auto for input and output.

When in input mode, the TCL I/O system translates the external end of line representatin into newline characters. And, when in output mode the TCL I/O system translates the newlines into external end of line representation.

The performance of the translation mode when specified as input and when specified as output is discussed below.

1. auto

When in input mode, it translates newline (*lf*), carriage return (*cr*) and *crlf* (carriage return followed by a newline) into end of line representation.

When in output mode it chooses different representations for different platform.

For example, for sockets on any platform, it uses *crlf*. On unix platform, it uses *lf*. On Macintosh platform, it uses *cr*. On windows, it uses *crlf*.

2. binary

In this mode, no translations are performed during input or output. This mode is much similar to *lf* mode.

3. cr

In this mode, the end of line is represented by a single carriage return character. When in input mode, it translates carriage returns into newline characters. And when in output mode, it performs reverse translation. It is used on Macintosh platforms.

4. crlf

In this mode, the end of line is represented by a carriage return character followed by a newline character. When in input mode, it translates the carriage return-newline characters into newline characters. And, when in output mode it performs reverse translation. It is use on windows platforms and for network connections.

5. lf

In this mode, the end of line is represented by a single newline character. No translations are performed during input or output. It is used on unix platforms.

5.12 Advance TCL

5.12.1 eval, source, exec and uplevel Commands

Q25. Explain the following commands,

(a) eval

(b) uplevel.

Answer :

(a) eval

The creation and execution of TCL script is done by the 'eval' command. This 'eval' command takes a number of arguments and concatenates them using a separator 'space'. It then calls the interpreter to execute the output (resulting string) as a TCL script. The most common use of 'eval' is that it allows all the TCL parsing rules to be applied to the script. By this, the script contains a large number of commands, comments, multiple lines etc. The commands generated by 'eval' are stored in variables and are later executed as TCL scripts.

Example

```
set clear
{
      set x 0
      set y 0
}
eval clear
```

When the command 'eval' is invoked, the variables in the script are set to '0'. The only use of 'eval' in the above example is that the two set commands are executed directly. The advantage of 'eval' is more useful in dynamic processing of creating the script.

Generally, when parsing a command the TCL allows two levels of parsing. One level of parsing and substitution is with the use of 'eval' command. Another level of parsing is done, where result of one substitution is provided (reparsed) to another substitution. This scenario is useful when a variable contains a list of items, or a return value with a list of items and when the values in a result are passed separately to the command.

Examples

```
set vars   {x y  z}
for each n $ vars
        {
              unset $n
        }
```

Here, the single unset command takes a number of arguments and tries to unset them as it is in the "foreachloop". Thus, the above code works five in the absence of 'foreach' loop the above code it does not work well.

1. Set vars{x y z}
  unset $ vars

Here, the variables are passed as single argument to the unset command. So 'unset' tries to unset a variable named as a b c d.

To make the above code work, treating, the variables as separate arguments the {∗} syntax is used for arguments as shown below,

2. Set vars{a b c d}
  unset {*}$ vars

The same thing can be done by using the 'eval' command, which creates a new form of a command, unset a b c d by concatenating all the arguments. This new command is finally passed to TCL for evaluation.

3. eval [concat unset $ vars]
  or
  eval unset $ vars

The two codes shown above are similar to each other. The variable names are treated as separate arguments to unset in both the cases.

(b) Uplevel

The 'uplevel' command is the combination of 'eval' and 'upvar' commands. This command allows a script to be interpreted in a specific context. A relative or absolute stack frame number used with uplevel denote the level similar to upvar and the default value is 1.

Q26. Explain the following commands,

(a) source

(b) exec.

Answer :

(a) source

This command takes the file name as an argument. It invokes the interpreter to execute the content of a file as TCL script. The syntax of this command is as follows,

```
source  ? – encoding  encoding  ? fileName
```

encoding in the syntax is used to specify the encoding. If encoding is not specified then TCL reads a file using system encoding.

Example

```
source  demo.tcl
```

The file can be specified either using relative or absolute path. The file demo.tcl is executed as a script. The return value of source command is the value returned from the last command in the file.

(b) exec

The 'exec' is implemented on systems that support process creation. The first argument specifies name of executable file. TCL searches for the matching file specified as argument and executes the file in a different child process and waits for its completion.

Example

```
set  today [exec  date]
```

The date command of the system is executed and the value returned by it is stored in the variable "today".

Note

This command is not confined to system commands.

5.13 Namespaces

Q27. Explain the autoload feature of TCL.

Answer : Model Paper-II, Q5(a)

The command 'unknown' is invoked by the TCL interpreter, when it finds a new command that does not exist. The name and arguments of the original command are passed to the unknown command.

Example

```
Set  a  20
create database  library  $ a
```

Here, the createdatabase command is used. But, if createdatabase command does not exist. We rewrite the same using the 'unknown' command as shown below,

```
unknown   createdata base   library  20
```

Before the invocation of the command 'unknown' as the substitution are performed.

The default version of 'unknown' command is that it performs the following operations as provided by the TCL system.

(a) If the procedure defined in a library file is the command, then it find out the source of procedure to re-invoke the command. This process is called 'autoloading'.

(b) If the command name is same as that of program name then 'exec' is used to execute the program. We call this as 'autoexec'. Any unix shell command can be used by the TCL script running under UNIX environment. Example, if we type '*ls*' command then the files in the current directory are listed by the 'unknown' command which invokes 'exec *ls*'. The commands such as standard input, standard output, and standard error are all to be re-directed to the corresponding TCL channels in the case if command cannot perform redirection. This behaviour of 'exec' is different from normal behaviour. It allows more and vi to be involved directly from a TCL application.

(c) If a special symbol example!! is included in place of the command name then the previous command is invoked again.

(d) If a command, which exists already has a unique abbreviation which is similar to the command name then after the expansion of abbreviated command name, the command is called again.

The autoloading process initially makes the 'unknown' command aware of the library files. If this is fulfilled then the 'unknown' command locates the library file, sources it and then re-invokes the original command. This is done when a library procedure is used for the first time. But, if the command is used later then autoloading is avoided as it will be defined.

Q28. Define package. Explain about the creation and usage of a package.

Answer :

Package

The mechanism of package is similar to the autoloader.

Some improvements are made in the autoloader to result packages.

A package may have any number of versions. The script can impose condition on the required version.

A package can posses different TCL procedure definitions and commands in *C*.

Creation of a Package

The script files are available in different directories similar to autoloading mechanism. The command used to implement a package in a file is 'package provide'.

Example

```
package provide utils 3.1
```

Here, the library file implements 'utils' package whose version is in the format 'major.minor' (i.e., Version 3.1). The change in the major version number indicates a significant change to the package whereas the changes in the minor version number indicates backward compatibility. The same package can be implemented by a number of library files. The concept of package is on abstract one. That is, it is placed on the top of a physical file structure.

pkg_mk Index is a TCL procedure which provides the list of procedures in a package. In the form of an Index similar to autoloader.

Example

```
pkg_mkInde x {c : \ tcl\ lib\ utils} *. tcl
```

The variable auto-path is set to list the directories that contains library files.

Usage of a Package

The command 'package require' is used to specify a package which is used by an application.

Example

```
package require utils
```

The procedures in a package are not loaded whenever the 'package require' command is invoked but as per the requirement of an application they are autoloaded, when called for the first time.

If the following is specified,

```
package require utils 3.1
```

Then the package versions preceeding the version '3.1' will not be used. If the package versions succeeding version 3.1 exits then they may be used.

Q29. Write short notes on namespaces.

Answer :

Apart from variables names, the TCL procedure names also occupy the global name space. For small scripts, the concept of global name space is sufficient. But, for large scripts, it is difficult to differentiate the names when using libraries. This problem can be solved by using the naming conventions provided prior to TCL 8.0. But, this does not provide a unique solution. The concept of namespaces available with TCL 8.0, provides separate memory contents for both the variable and procedure names within the global scope. This avoids the difficulty of differentiating the names when using libraries.

Using namespaces

The 'namespace eval' command is used to set and occupy a namespace.

Example

```
set a 0
namespace eval utils
{
        variable a
          ==
        set a 2
          ==
}
```

The first argument which is followed by the 'namespace eval' command recognizes the namespace. In other words, the second argument is the script to be evaluated in the recognized namespace.

The 'namespace eval' command creates a new namespace if a namespace does not exists.

In the above example, the namespace block contains '*a*' defined by the command 'variable'. It is different from the '*a*' defined outside the namespace. This is because a namespace creates new naming context.

The notation for referring the variables inside/outside a namespace is similar to the way in which pathnames are referenced in a hierarchial file system. The : : a is used to refer the outer '*a*' inside the namespace whereas utils : : *a* is used to refer inner '*a*' outside the namespace. Here, : : a and : : utils : : a are absolute names and utils : : a is a relative name. These names are determined from the global nameSpace, in which the nameSpace utils is nested inside it as shown below.

```
set a 0
namespace   eval   utils
{
        variable a
          ==
        set a 2
        namespace eval file_utils
        {
        variable a
          ==
        set a 4
        }
}
```

The absolute name : : utils : : file_utils : : a can refer the innermost 'a'. The relative name file_utils : : a is used to access 'a' from the namespace utils.

In a nameSpace, if a variable is declared using 'variable' command, outside a procedure then it becomes global to all the procedures in that nameSpace.

Example

```
namespace    eval  utils
    {
      variable   a 0
            ......
    }
```

The variable '*a*' has a initial value '0' which is used by all the procedures in 'utils' namespace. In other words, the command variable is also used to initialize the variable with initial values.

In the same way, the variable command when used inside a procedure, a global variable in a namespace can be accessed in a procedure body which resembles global command outside the namespace.

Example

```
namespace eval utils
{
        Variable a 0  b 1
        proc  demo {a}
        {
              variable b
                ==
        }
}
```

Q30. Explain the commands for importing and exporting namespaces.

Answer :

Exporting Namespace

The 'namespace export, command is used to access the library procedures without the need of relative (fully qualified) names. The export command is defined before the procedures in the name space. This command lists the procedures names that can be seen outside a namespace. By default the global namespace does not export any command.

Example

To Illustrate Export Command

```
namespace eval utils
      {
            variable a 0
            namespace export x1  x2
            proc  x1{x  y}
            {
                  variable a
                  ===

            }
            proc  x2 {x}
            {
            variable a
                  ===

            }
```

Importing namespaces

The commands that are exported can also be imported using the 'namespace import' command. By default, this command do not override any of the previously imported commands. In case the clash between an imported procedure name and a local procedure occurs then an error is raised. To explicitly override the existing commands force option is used. To delete the previously imported commands other than the existing commands 'namespace forget' command is used.

```
namespace   import  demo : : x1   demo : : x2
                        or
      namespace  import   demo  : : *
```

Note

In the global namespace, the importing and exporting of commands by library packages is not allowed.

Q31. Discuss about call backs.

Answer :

Call Backs

The script that are provided to an event driven application, will execute it whenever the application encounters an event such as key press or mouse click. These callbacks are executed by default in a global scope. To confirm that these call backs are evaluated in a specific namespace context, the command 'namespace code' is used. A button is defined as follows,

```
button .B1  – command {puts  $a}
```

When the button is pressed its associated script gets executed.

If the above command with variable '*a*' is encountered in a namespace then it can be clearly stated that the variable 'a' belongs to the namespace and not the global scope by modifying the above command as follows,

```
button .B1  – command   [namespace  code {puts  $a}]
```

5.14 Trapping Errors

Q32. Write briefly about trapping errors.

Answer :

Generally, errors in a script makes the TCL interpreter to abort all the active TCL commands. Sometimes, it is necessary to execute the entire script even when the script contains errors. For this purpose, we use the command "catch". This command ignores the error statement and allows the interpreter to execute the remaining script. The usage of catch command is shown below,

```
catch   {Script}
```

The catch command performs error trapping. It returns '0' if the execution of script is completed normally, otherwise, it returns '1' indicating error condition. The script is provided as an argument to the catch command, which evaluates the script.

To understand the usage of 'catch' command consider an example in which a process tries to open a file and read the data. To check whether a particular file exists and the process has access to the file (for reading), the TCL provides file readable command.

Though, the file exists, there are chances that another process may remove it before the user gets a chance to open and read it.

Example

open demo.txt

ϕ could not open "demo.txt" : errormessage

no such file

To overcome this problem catch is used. It ignores the errors in open command by writing as follows,

catch {open demo.txt}

⇒ 1

The return value of catch determines the success/ failure of open command. Since, the return value is 1. It indicates failure. Another form of 'catch' contains the second argument as shown below,

catch {script} variable

The second argument specifies the name of the variable. It is used to store either the return value or the error message.

catch {open demo.txt} pseudo

⇒ 1

set pseudo

⇒ could not open "demo.txt" : no such file

If the open command terminates successfully then 'pseudo' would contain '0' as the return value indicating success. In this case it holds an error message as the 'open' fails to terminates successfully.

5.15 Event-driven Programs

Q33. Explain the event-mode of TCL. Also explain event-handling of various events.

Answer : **Model Paper-I, Q5(b)**

TCL is basically designed as a command_line system. But, at present it supports event driven programming. The event model of TCL can be easily understood as the event and their associated commands are registered by a script. If an active event loop is available then, whenever an event is encountered, the commands are executed by the system.

The 'Vwait' command is used to control the event loop.

Example

set a 0

Vwait a

The variable 'a' acts as argument to the command Vwait. This is the starting point of the event loop and the event process continues till the value of 'a' is changed by some event handler. Finally the Vwait command returns.

There are three classes of events. They are as follows:

1. Timer events
2. Idle events
3. File events.

1. Timer Events

These events are raised after the specified time is completed. The command 'after' registers timer events.

Example

after 250 [list proc arg1 arg2]

After 250 milliseconds the procedure 'proc' is called with arg1 and arg2.

after 500

This has only one argument which indicates the application to be paused for specified time (i.e., 500 milliseconds).

The value returned by 'after' command is used to identify the event.

Example

set = sum [after 500 [list proc arg1 arg2]]

This identifier cancels the pending timer events. The variant of after command cancels the pending events as shown below,

Example

after cancel sum.

2. Idle Events

These events are self explanatory events. Whenever the system has nothing to do, these events are raised. If the first argument of 'after' command is idle, then the command is executed indicating idle moment.

3. File Events

To register the file events, the command 'file event' is used. The read and write handlers for a channel are specified by this command. These handlerss are called whenever the channel is opened for reading or writing. It is mostly used with pipes and network sockets. In a pipe, if there is no data available to read, then the request blocks. To avoid the blocking of a request we use file events.

Example

```
set n [open "\ cmd 1"]
# Read handler is registered
file event $n readable [list read_handler $n]
........
```

Read Event Handler

This eventhandler uses 'gets' to read one_line at a time inorder to overcome the danger of blocking when there is no data available in the pipe. The handler will be called immediately after its existence, when not more than one line or block is available.

As soon as, end of file is reached the channel gets ready to read the data, which causes the handler to be called. So, end of file with in the handler is checked and the channel is closed. This automatically unregisters the handler.

The following code shows, how a read_handler is called repeatedly which is previously registered and not checked for end of file.

```
proc read_handler {pipe}
{
        global line
        if[e of $pipe]
        {
               close $ pipe
               return
        }
        #reads one line at a time.
        gets $pipe line
               ==;

}
```

Sometimes, the close operation on a handle may fail. Thus, it is explicitly closed as shown below,

```
if [e of $pipe]
{
        if[catch {close $pipe}]
        {
        fileevent $pipe readable " "
        }
        return
}
```

Here, empty string is registered as the newhandler to cancel the existing handler.

Non-blocking I/Oc

To read a pipe we can set event driven model. However, there is a possibility of an application being blocked. If there is some data available in pipe then the event 'pipe readable' is generated.

The "gets" in read_handler may block if the pipe has only a partial line. In order to escape from this situation the pipe is set to 'non-blocking' mode as shown below,

```
set a [open "\cmd 1"]
fconfigure $a -blocking 0
#  read handler is registered
 ==
```

When there is no data available in the pipe and when 'gets' is called on a non-blocking channel, it will return-1. Here, the read-handler should be altered in such a way that it does not perform any thing when '–1' is returned (i.e., simply ignore the case).

Only in a non-blocking channel there is a possibility of blocking write operation. But, this "situation", rarely occurs.

5.16 Making Applications 'Internet-Aware'

Q34. Discuss about making applications internet aware.

Answer :

The data stored on different servers such as POP/IMAP mail servers, FTP servers, Newservers, Webservers etc., is available through internet. Thus, internet becomes rich source of information. There are many ways to access the information webserver and FTP server. One such way is by using the Web browser. Similarly, information on Newsserver and mailserver can be accessed using expert clients. The application that can access information without manual involvement of a server is 'Internet-ware'.

Example

The look-up facility provided by a website takes the query entered by a user when submit button is clicked. This query is then passed to CGI program on the server, which resolves it and sends the response in the form of a webpage to the browser. In the format specified by a browser the request is sent to the server, this is done by a TCL application which makes a connection to the server. The response from the server which is in HTML form is gathered by the application to extract the useful fields for providing answer to the query. Similarly, a TCL application can setup connection to a pop3 mailserver and send a request. In this case the server returns the number of unread messages in the form of lists.

Scripting languages hide the complexity of operations especially when using libraries and packages. Thus, they are becoming powerful. In these languages only few lines of code is sufficient to complete a task whereas languages like 'C' requires pages of code. The above described interaction is unimportant when considering the HTTP packages available with TCL distribution. The usage of HTTP package discussed below,

The 'HTTP' Package

It is not an is easy task to retrieve a webpage from any server. The following code retrieves a webpage i.e., home page from the server www.oxygene.com

```
package require http 1.0
set token [: : http : : geturl
                 www.oxygene.com]
```

The token that identifies a state array is the value of the 'token'. This state array contains the result of the transaction.

```
upvar # 0 $token state.
```

The above command converts a token into an array. The page body is available as state. The elements that a state array possess are as follows:

1. **current Size**

   This element gives the number of bytes sent.

2. **http**

   This element gives the reply status of HTTP.

3. **error**

   This element displays the error code if a transaction is aborted.

4. **url**

   This element provides the requested URL.

5. **Status**

   This element can be OK, end of file (eof) or reset

6. **meta**

   This element consists of a list of {key value} pairs in reply header which are extracted from <META> tags.

7. **type**

   This element returns the type of content

8. **totalsize**

   This element returns the total size of the data provided.

The above 8 elements of a state array can be retrieved using individual functions given below,

(i) **: : http : : data $ token**

The URL data of the state array is returned

(ii) **: : http : : status $ token**

The status element of the state array is returned.

(iii) **: : http : : code $ token**

The http element of the state array is returned.

(iv) **: : http : : size $ token**

The current size element of state array is returned.

5.17 Nuts and Bolts Internet Programming

Q35. Define sockets, servers and server sockets in TCL.

Answer :

Sockets

Originally, sockets were a feature of Berkeley UNIX. Later, they were adopted by UNIX system. Thus, it became the defacto to mechanism of Network communication channel.

Socket is a network communication channel that provides two way communication between the processes on different machines.

Example

The window 9x and winsock NT4 have built in sockets. The winsock package provides window the similar functionally. It allows the window system to communicate with UNIX system over the network.

Servers

Server is continuous process that manages access to various resources.

Example

The HTTP server provide access to web pages and the FTP servers retrieve the file using file transfer protocol.

Every socket contains a port number by which server-client connection is established.

Server establishes connection on well known ports which is specified by the client during socket creation.

Example

The port number 80 is used by the client to communicate with a web server.

Server Socket

Server socket provides access to multiple connections.

When a connection is established with the well known port, then a new socket is created and the server listens to the further requests.

Server sockets are well known as listening socket. Server sockets are created by using socket command with server option as shown below,

```
socket – server command ? options ? port
```

This will establish the server socket on the current host at a given port number. Servername can also be passed instead of port number depending on the operating system.

Q36. Explain about the creation and usage of sockets.

Answer :

Socket is created using the socket command. It is a network communication channel that provides two way communication between the processes on different machines (i.e., the client and the server)

Syntax of Socket Command

To set up a socket, its first argument should be domain name or host name, even numeric IP address is acceptable. And the second argument should be port number.

TCL sets the socket using a single statement, but programming languages like C needs thirty lines of code to set up a socket.

Example

```
set mysocket [socket localhost 8080]
```

Above statement will open a client side socket on the local system whose port number is 8080. It returns a channel identifier which is to be stored in a socket.

The socket programming can be clearly explained using a simple script to access a POP3 server and find if there is any mail waiting.

By simple client_server interaction POP3 client will retrieves the mail from the POP3 server.

The conversation between the client and the server is discussed below,

Client : establish connection

Server : + OK POP3 penelope V 6.50 serer ready

Server will wait for the username. If the given user name its accepted, it will prompt for password.

Client : USER xyz

Server : + OK User name is accepted, passwordplease

Now, the server waits for the password. If the password entered is correct, then the server will respond by displaying the number of messages new present.

Client : PASS 12346

Server : + OK Mailbox open, 2 messages

Note

If the operations performed are successful, the message will always start with +OK.

Q37. What are the problems with network programming? Suggest solution to overcome these problems.

Answer :

The problems associated with network programming are as follows:

1. Failed connections
2. Connection timeout
3. Non-blocking I/O.

1. Fail Connection

This problem arises when a socket command fails to connect the client and the server.

Example

If the host specified does not exist, then the exception will be handled by catch.

```
if [catch {socket www.xyz.com 8080} status]
{
    puts stderr $ status
}
    else
{
    set  mysocket $ status
}
```

2. Connection Timeout

If the server is busy, then the request made by the user is put in a queue and is allowed to wait in the queue till the server accepts the request or refuses the request.

– async option

If the socket uses –async option, then it immediately returns even if the established connection fails. Hence, the socket becomes writable. A fileevent is used to recognise this, is as follows,

```
set mysocket [socket –async  www.xyz.com  8080]
fileevent $sock writable {set connected 1}
Vwait connected.
```

Until the fileevent is set to value connected 1, Vwait blocks all other requests and runs the event continuously in a loop.

The value of the variable connected will be changed after certain time using a timer event in low cunning. This is discussed in the example below,

Example

```
after 12000 {set connected 0}; # 3 min
set mysocket [socket - async www.xyz.com8080]
fileevent $mysocket writable {set connected 1}
Vwait connected
if ! connected
{
    puts stderr "server connection timeout"}
    exit
}
```

If the connection established is timeout, it will exit running the request.

Catch is used to handle the exception, when a wrong address is requested.

Example

```
after 12000 {set connected 0}; # 3 min
if [catch {socket –asyne www.xyz.com 8080} status]
puts "connection failed"
puts "$ status"
exit
}
else
{
        set mysocket $status
        fileevent $mysocket writable {set connection 1}
        V wait connected
        if ! connected
        {
               puts stderr "server Not responding"
               exit
        }
}
```

3. Non–Blocking

Network connection are delayed by keeping the request in the queue. Until the server respond's to the request, the request should not freeze. This is achieved by setting the socket channel to non-blocking mode.

5.18 Security Issues

Q38. Explain slave interpreter and safe interpreter.

Answer :

Slave Interpreter

A slave interpreter is a TCL interpreter that run as a separate process under the control of master interpreter.

A slave interpreter creates a new instance of TCL interpreter that has its own global and procedure namespace. It does not have privileges to access the variables or procedures of the master with an exception of sharing only environment variable is used between the master and the slave.

Creation

A slave interpreter is created using the interp command. It is followed by create and name of the interpreter i.e., demointerp.

```
interp create demointerp
```

Now, demointerp is run as a separate process. It waits for the script to interrupt so that it gets evaluated using.

```
interp eval demointerp {puts "Hello this scan\n"}
```

This can also be written as,

```
demointerp eval {puts "Hello!, this is slave \n"}
```

This eval command will return the value of the last expression evaluated in the slave that provides communication between master and slave. If the current script evaluation is completed slave waits for another script. This process is repeated until the slave is killed by master using interp delete command as follows,

```
interp delete demointerp
```

Safe Interpreter

The safe interpreter is a mechanism which handles the situations like parsing of TCL scripts which are embedded within a web page, e-mail message or in some content where code creator is not known.

The safe interpreters are nothing but slave interpreters that have a special flag set during creation using_safe option. This is shown below,

```
interp  create_safe  demosafe
```

It hides all the commands such as exec, open etc., which are made accessable only when master interpreter makes a explicit call to them.

invokehidden Command

This command is used by the master interpreter to invoke the hidden commands. They implement a safer version of the source file, checks whether it is a TCL script file in the TCL library or not by using the source command as shown below,

```
interp  invokehidd en  demosafe  source   $ file
```

The interp expose command is used to make a hidden command permanently visible.

```
interp  expose  demosafe  pwd
```

Here, the pwd command is made visible to demosafe. A master interpreter can even hide the commands other than the default hidden command list by using hte hide command.

```
interp  hide  demosafe  after
```

Here, the after command is hidden in demosafe interpreter.

If a particular command has to be removed from the interpreter entirely we use eval as shown below,

```
interp  eval  demosafe  [list  rename  after{ }]
```

Here, the after command is removed from the demosafe interpreter.

Q39. What is meant by safe base?

Answer :

The safe base implementation is done : : in safe module. It provides privileges to the master interpreter to create safe interpreter. The safe interpreter consists of all the predefined aliases like source, load, file and exit commands. The autoloading and the package mechanisms are enabled for this use.

Safename

The safe namespace contains all the commands provided by the safe base in the master interpreter.

Source and Load

The source and load aliases can be done on files that exist in the list of directories that are provided by the master when safe interpreter is created.

Exit

The exit alias terminates the slave interpreter.

File

The file alias provides safe access to the sub commands of the file command. Since, the local file system is accessed by source, load and file commands, the information is leaked. That is, the directory structure is made disclosed. Inorder to avoid this information leakage, tokens are used instead of file name as an argument to the commands in the safe interpreter. These tokens are translated to the real directory name.

Example

Master interpreter mediates the source file. It has the virtual path system for each safe interpreter that maps the token accessibility into a real path name on the local file system. This prevents the safe interpreter from gaining knowledge about the file system structure of the host which is executing the interpreter.

Q40. What are the two ways of passing file descriptors between masters and slave?

Answer :

The file descriptors are shared between the slave and the master so that the file is accessed by both of them. The two ways of passing file descriptors between masters and slave are as follows:

1. Using Share Command

The first method to share the file descriptor between masters and slaves is using the share command. The master opens a file for writing. Using

```
set file 1 [open / xyz /demo] w 0666
```

Now, the file descriptor is shared by the slave using

```
interp share { } $file 1 demo
```

Here, { } represents the current interpreter. And, the channel gets closed only after both the interpreters reach the close command either implicitly or explicitly. The I/o channels accessibility will also be automatically closed in an interpreter as soon as it gets destroyed

2. Using Transfer Command

Alternative method to share the file descriptor between masters and slaves is by using transfer command.

```
interp transfer { } $file 1 demo
```

This closes the master interpreter implicitly and transfers the file descriptor to the slave interpreter. To have complete control over the file that is transferred, alias command is combined with transfer command.

Example

If we want demo to be accessed for reading but not for writing the open command is aliased in the master. This will set the file to read-only mode before opening it and then transfer the file descriptor. This is explained in the following code,

```
proc safeopen {path mode permission}
{
global demo
// sets read only permission to the file demo
setchan [open $ path r $ permissions]
// Transfers the file
interp transfer { } $ t demo
return $t
}
demo alias open safe open.
```

5.19 C Interface

Q41. Explain about integrating TCL into an existing application.

Answer : Model Paper-I, Q5(a)

To integrate TCL into an application, only a TCL header is included and TCL_create interp must be called.

```
# include < tcl.h>
Tcl_interp * int p
...
intp = TCL_CreateInterp( )
```

Above function will return a pointer to TCL_Interp structure. This return value will be passed as an arguments to other function to identify the interpreter that is involved. As, a single application can invoke multiple interpreters.

Calling the Interpreter in the Earlier Versions of TCL i.e., Prior to TCL 8.0

TCL_Eval can be invoked as soon as interpreter is set/ up

Example

```
char scpt[ ] = "puts {welcome students}";
result = TCL_eval (intp, scpt);
```

If the interpretation is successful, then the result variable is assigned with TCL_OK. Otherwise, the result variable is set to TCL_ERROR.

The intep → result has the pointer to the resultant string.

There are several variants on TCL Eval which are listed below.

1. TCL-Evalfile : The second argument of TCL-evalfile is filename. If treats the contents of file as scripts.
2. TCL-VarEval : An arbitrary number of string arguments are followed by the interpreter argument. It concatenates all strings into one string in order to evaluate it as a script. The last argument TCL_VarEval must always end with NULL i.e., as a last argument.

It stores the internal values like reference count type, string value, length and internal representation as fields of the structure. The TCL_obj object stores a string that represents a decimal number in the string field when used in a numerical context type conversion takes place (i.e., from string to number) And, the resultant binary number is stored in internal representation field.

A technique of compiling scripts into a byte code format was also introduced in TCL 8.0. Here, the TCL_evalfile is replaced by TCL_EvalobjEx. The second argument is the pointer to TCL_obj structure rather than a string as in TCL_Eval function.

When the script is used for the first time it will be stored in string field. After the script is compiled into byte code format it is stored in, internal representation field. When TCL_Evalobj Ex function is called again then bytecode version of the script will be used. It returns TCL_OK or TCL_ERROR.

It stores the resultant string or error message in the TCL_obj structure which can be later retrieved using TCL_Getobj Result.

Q42. Explain about creating new commands in C.

Answer :

The structure of a command procedure prior to TCL 8.0 is discussed below.

When the TCL interpreter recognizes a C coded command while passing then these argument words must be passed to the C program. This is down in the same way as shell does to pass arguments to a C command.

In C, the command line starts with the following line.

```
main(int argc, char *argv[ ])
```

Here, argc is the count of number of arguments and argv [ ] is the pointer variable that holds the address of array of string.

The function will be called using value of argv[0] as its name. The TCL procedure for the above line is as follows,

```
type def int TCL_cmd Proc (client Data client,
        TCL_Inter *intp,
        int argc,
        char *argv[0]);
```

The argc and argv in the above prototype will behave in the same way as in main function.

The second argument is the pointer value to the interpreter.

This TCL command procedure returns a value, TCL_OK or TCL_ERROR

The return value is stored in interp → result similar to TCL_Eval

Now, let us discuss the structure of TCL command procedure inTCL 8.0

In TCL 8.0, TCL_obj structure hold all the values. Thus, an array of pointers to these structures are passed as arguments to command procedure rather than the array of pointers to string. The prototype of the procedure in TCL 8.0 is as follows,

```
(type def int TCL_Ocmdproc(clientData, client,
        TCL_Interp *intp
        int obj1,
        TCL_Obj *CONST obj2[ ]);
```

This procedure also returns TCL_OK or TCL_ERROR. The TCL_set obj result is used to set the result as a TCL_obj structure.

This result can be later retrieved using TCL_GetObjResult.

Registering a new Command in the Version Prior to TCL 8.0

The TCL create command function is used to register a new command with the interpreter. This command takes 5 arguments. They are as follows,

1. A pointer to the interpreter. This is used to register the command. This pointer is passed to the command procedure when it is called.
2. The command's name that is used in TCL.
3. The C command procedure's name that implements the command.
4. The client data i.e., client. It is passed as first argument to the command procedure when it is called.

   The client data argument is used if a single command procedure is used to implement a number of commands. Each command is registered to identify that it belongs to particular family.
5. The 'delete' procedure.

   The delete procedure is called when the TCL command is deleted using TCL_Delete command.

   Example: TCL_createCommand intp, "wibble". wibble Cmd(ind, (client)NULL,

   (TCL_(CmdDeleteProc*)NULL);

   Here, the TCL command "wibble" is added. And, a command procedure inc is written to implement it.

Registering a New Command in the Latest TLC Version 8.0

The registration of a new command and in TCL 8.0 is similar to the registration in previous versions except that it is registered using TCL_createObjCommand rather than the TCL_createCommand.

Q43. Write short notes on dynamically loadable packages.

Answer :

In C, packages are defined by creating a shared library which is loaded dynamically.

For Example

In response to a package require command.

To create a package named 'wib' that provides TCL command 'makewib' the following code is written.

```
/*wib.c*/
# include < tcl.h>
/*Declaring command procedures and functions*/
int makewibcmb (clientData cd,
TCLInterp* tip, int obj1,
TCL_obj * CONST obj2[ ]);
/*loading the package*/
int wib_Init (TCLInterp*tip)
{
TCL_createObjCommand (tip,"wib_a",
makewibcmb, (cd)NULL,
(TCL_cmdDeleteproc*)NULL);
TCL_Pkg (tip, "wib", 1.0);
return TCL_OK;
}
```

After compilation, the compiled package is placed in the directory, Now,

Pkg_mkIndex is run in the directory.

The index entries will be created. When the compiler reads the following,

package require wib.

This calls the wib_Int function for creating a new command.

Finally, the TCL_PkgProvide function will create the Pkg Index.tcl file. This file in referred by the autoleader when a command defined in this package is used for the first time.

OBJECTIVE TYPE

I. Fill in the Blanks

1. ____________ operator provides a conditional expression.
2. The command that return the number of elements in a list is, ____________.
3. The stream used to output a string is ____________.
4. The command which calls the interpreters to execute the contents of the file as TCl script is, ____________.
5. The command which combines eval and upval is ____________.
6. ____________ are executed by default in the global scope.
7. Error trapping is allowed by the ____________ command.
8. The events which are raised after a specified time has elasped are called ____________ events.
9. The function used to register a new command with the interpreter is ____________.
10. The slave interpreters are created and controlled using the ____________ command.

II. Multiple Choice

1. Which of the following is not an input command? [ ]

   (a) Gets (b) Read
   (c) Puts (d) None of the above

2. Which of the following is used as the last entry in the argument list for defining a variadic procedure? [ ]

   (a) Default (b) Args
   (c) 1 (d) 0

3. Which of the following pattern is not used for Glob matching? [ ]

   (a) * (b) ?
   (c) [ ] (d) .

4. The command used to control the event loop is, [ ]

   (a) Vwait (b) Set
   (c) After (d) After cancel

5. Which of the following command allows us to treat a list as an array? [ ]

   (a) Lindex (b) Lrange
   (c) Lreplace (d) All the above

6. Which of the following command provides variable and command substitution? [ ]

   (a) Expression (b) Pattern
   (c) Expr (d) Glob

7. Which of the following is useful for tidying up the string? [ ]

   (a) String trim (b) String trim left
   (c) String trim right (d) All the above

8. The choice operator is represented by ____________ [ ]

   (a) ? (b) Switch case

   (c) ? : (d) ||

9. In how many ways the file descripters are passed between the master and the slave? [ ]

   (a) 1 (b) 2

   (c) 3 (d) 4

10. Which of the following network communication channel provides a bidirectional flow between the processes on different machines? [ ]

   (a) pipes (b) Fifos

   (c) Sockets (d) Both (a) and (b)

K EY

I. Fill in the Blanks

1. Choice
2. Llength
3. Stdout
4. Source
5. Uplevel
6. Call backs
7. Catch
8. Timer
9. TCL _ create command
10. Interp

II. Multiple Choice

1. (c)
2. (b)
3. (d)
4. (a)
5. (d)
6. (c)
7. (d)
8. (a)
9. (b)
10. (c)