**TCL Syntax**

<program> -> <stmnts>

<stmnts> -> <stmnt>; <stmnts>|

<stmnt> NEWLINE<stmnts>|

E

<stmnt> -> <seq\_stmnt> |

<cond\_stmnt> |

<loop\_stmnt> |

<funct\_def> |

<funct\_call>

Sequential Statements:

<seq\_stmnt> -> <assign\_stmnt>|

<cmnt > |

<expression>

Assignment Statements:

<assign\_stmnt> -> <key>set <vars>

<key> -> <key\_word> | E

<key\_word> -> list | dict

<vars> -> <var1> <var2> <vars> | E

<var1> -> <simp\_word>|<Array\_var>

<var2> -> $<word>|<word>|<Array\_var>|<expression>

<Array\_var> -> <word> (< word> )

Comments:

<cmnt> -> # <words>

Expressions:

<expression> -> expr <exp> | <exp>

<exp> -> <term1> ? <term1> : <exp> |

<term1>

<term1> -> <term1> || <term2> |

<term2>

<term2> -> <term2> && <term3> |

<term3>

<term3> -> <term3> | <term4> |

<term4>

<term4> -> <term4> ^ <term5> |

<term5>

<term5> -> <term5> & <term6> |

<term6>

<term6> -> <term6> in <term7>|

<term6> ni <term7> |

<term7>

<term7> -> <term7> eq <term8>|

<term7> ne<term8> |

<term8>

<term8> -> <term8> == <term9> |

<term8> != <term9> |

<term9>

<term9> -> <term9> < <term10> |

<term9> <= <term10> |

<term9> >= <term10> |

<term9> > <term10> |

<term10>

<term10> -> <term10> << <term11> |

<term10> >> <term11> |

<term11>

<term11> -> <term11> + <term12> |

<term11> - <term12> |

<term12>

<term12> -> <term12> \* <term13> |

<term12> / <term13> |

<term12> % <term13> |

<term13>

<term13> -> <term13> \*\* <term14> |

<term14>

<term14> -> + <term15> |

- <term15> |

! <term15> |

<term15>

<term15> -> [<expression>] |

<number>|

$<simp\_word> |

<funct\_call>

Conditional Statements:

<cond\_stmnt> -> <if\_stmnt> | <switch\_stmnt>

<if\_stmnt> -> <if\_body> <elseifs> <else>

<elseifs> -> <elseif\_body> <elseifs> | E

<else> -> <else\_body> | E

<if\_body> -> if <expression> then <body>

<elseif\_body> -> elseif <expression> then <body>

<else\_body> -> else <body>

<switch\_smnt> -> switch <expression> { <cases> } |

switch <expression> <cases>

<cases> -> <case> | <cases> | <cases> <def\_case>

<case> -> <word> { <body> } |

“<word>” “<body>”

<def\_case> -> default { <body> } |

“default” “<body>”

<body> -> <stmnt> | { <stmnts> }

Loop Statements:

<loop\_stmnt> -> <while\_stmnt> | <for\_stmnt>

<while\_stmnt> -> while <test> <body>

<test> -> “ <expression>” | { <expression> }

<body> -> <stmnt> | { <stmnts> }

<for\_smnt> -> for <start> <test> <next> <body>

<start> -> { <assign\_stmnt> }

<test> -> { <expression> }

<next> -> { <stmnt> }

<body> -> <stmnt> | { <stmnts> }

Function Definitions:

<funct\_def> -> proc <simp\_word>{ <args>} {<body>}

<body> -> <stmnts> | <stmnts> return <word>

<args> -> <arg> <args>|E

<arg> -> <simp\_word> | <def\_arg>

<def\_arg> -> {<simp\_word> <word>}

Function Calls:

<funct\_call> -> <key><simp\_word> <params>

<key> -> <key\_word> | E

<key\_word> -> list | dict

<params> -> <param> <params> | E

<param> -> <word>

Words:

<words> -> <word> |

<words>

<word> -> <simp\_word>|

<quoted\_word>|

<bracket\_word>|

<substi\_word>|

<expansion\_word>|

[<expresion>] |

[<funct\_call>]

<number> -> <digit><number>|<digit>

<digit> -> [0-9]

<simp\_word> -> <first\_letter> <letters>

<first\_letter> -> [A-Z] | [a-z] | \_

<letters> -> <letter> <letters>

<letter> -> [A-Z] | [a-z] | \_ | $ | [0-9]

<quoted\_word> -> “ <words> “

<bracket\_word> -> { <words> }

<substi\_word> -> [ <words> ]

<expansion\_word> -> {\*} <words>