TABLE OF CONTENTS

Contents ***Page***

PROGRAM STRUCTURE

START = VIPER HEAD MAIN IS STMT END FUNCTION TAIL

FILE EXTENSTION

Viper compiles any file that uses the file extension **.vpr**

PARSING ALGORITHM

Viper Programming Language utilizes the **PREDICTIVE PARSING ALGORITHM** for parsing the input.

Below is the predictive parsing algorithm:

```
Initialize STACK <S, $>
      where S = start symbol
            $ = stack bottom marker
Then repeat:
      case STACK of:
            <X, rest> : if T[X, *next] = Y1 … Yn then
                              STACK ← <Y1 … Yn rest>;
                        else error();
            <t, rest> : if t == *next++ then
                              STACK ← <rest>;
                        else error();
until STACK is empty
```

The algorithm states that first, the stack must be first initialized (where the bottom of the stack is marked with the $ sign. Then the following steps are repeated until the stack is empty:

1. If the top of the stack, X, is a nonterminal symbol:
    a. The table function will X and the current token being read by the parser.
    b. Then return the derivation/s of the top non terminal symbol.
    c. Pop the stack.
    d. Push derivation/s from X. Otherwise, produce an error.

2. If the top of the stack contains a terminal symbol:
    a. Check if the current token being read by the parser is the same as the top of the stack
        i. If equal, then pop the terminal
        ii. Move to the next token. Otherwise, produce an error.

GRAMMAR 1: ORIGINAL GRAMMAR SPECIFICATIONS

CFG A = (V, T, P, S) where:

Variables (V)

```
V = { START, VIPER, HEAD, MAIN, IS, END, TAIL,
      FUNCTION, FUNCTIONS, ID, VAR_ID, FN_ID, FN_ARG, FN_ARGS, FN_CALL,
            PARAM, PARAMS,
      LET, ARRAY, OPT_RANGE, RANGE, RET, IN, OUT, FOR, INC_DEC,
      TO, DOWNTO, FOR, FOR_ARG, WHILE, DO, IF, ELSIF, ELSE,
      AND, OR, NOT, EQUALITY, EQUAL_TO, NOTEQUAL, GRTR_THAN, LESS_THAN,
      GRTR_THAN_OR_EQ, LESS_THAN_OR_EQ, PLUS, MINUS, MULTI, DIV, INT_DIV,
            REAL_DIV, MOD, EXP, PLUSPLUS, SUBTSUBT
      L_BRAC, R_BRAC, L_PAREN, R_PAREN, COLON, COMMA, SCOLON, DOT, DBL_QUOTE,
      SNGL_QUOTE, L_BRC, R_BRC, EQUALS,
      DATA_TYPE, INT, CHAR, CHARS, REAL, BOOL, VOID,
      STMT, DECLARATION, INPUT_STMT, OUTPUT_STMT, ASSIGN_STMT, LOOP_STMT,
            IF_STMT, FOR_LOOP, WHILE_LOOP, DO_LOOP, IF_PART, ELSIF_PART,
            ELSE_PART, OUT_ARG, OUT_ARGS
      LITERAL, INT_LIT, STR_LIT, CHAR_LIT, REAL_LIT, BOOL_LIT, NUMBER,
            NUMBERS, EMPTY,
      LETTER, LETTERS, TRUE, FALSE
      SIMPLE_DEC, INIT_DEC, CONST_DEC, ARRAY_DEC,
      EXPR, BOOL_EXPR, AR_EXPR, REL_EXPR, LOGIC_EXPR, LOGIC_AND, LOGIC_OR,
            LOGIC_NOT,
      LOG_REL, REL_TERM, REL_FACTOR, AR_TERM, AR_FACTOR, EXPONENT, AR_BASE,
            AR_EXP, STMTS}
```

Terminals (T)

```
T = { 'viper', 'head', 'main', 'is', 'end', 'tail',
      'let', 'array', 'range', 'returns', 'in', 'out', 'for', 'to', 'downto',
      'for', 'while', 'do', 'if', 'elsif', 'else', '&&', '||', '!', '==',
      '!=' '>', '<', '>=', '>=', '+', '-', '*', '/' '//','%', '^', '++', '--'
      '[', ']', '(', ')', ':', ',', ';', '.', '"', ''', '{', '}', '=',
      'integer', 'char', 'chars', 'real', 'boolean', 'void', 'EMPTY', 'true',
                  'false'}
```

Start Symbol (S)

S = START

Productions (P)

```
// start production
      START = VIPER HEAD MAIN IS STMT END FUNCTION TAIL

      VIPER = ‘viper’
      HEAD = ‘head’
      MAIN = ‘main’
      IS = ‘is’
      END = ‘end’
      TAIL = ‘tail’

// identifier
      ID = VAR_ID | FN_ID

// function call
      FN_CALL = FN_ID L_PAREN FN_ARG R_PAREN SCOLON STMTS
      FN_ARG = VAR_ID FN_ARGS | NULL
      FN_ARGS = COMMA FN_ARG | NULL

// function definition
      FUNCTION = FN_ID L_PAREN PARAM R_PAREN RET FN_RET IS STMT END FUNCTIONS
                  | EMPTY
      FUNCTIONS = FUNCTION | NULL
      FN_RET = DATA_TYPE | VOID
      RET = ‘returns’
      VOID = ‘void’

// function parameters
      PARAM = VAR_ID COLON DATA_TYPE PARAMS | NULL
      PARAMS = COMMA PARAM | NULL

// data types
      DATA_TYPE = INT | CHAR | CHARS | REAL | BOOL
      INT = ‘integer’
      CHAR = ‘char’
      CHARS = ‘chars’
      REAL = ‘real’
      BOOL = ‘boolean’

      L_PAREN = ‘(’
      R_PAREN = ‘)’
      COLON = ‘:’
      COMMA = ‘,’

// statements
      STMT = DECLARATION | INPUT_STMT | OUTPUT_STMT | ASSIGN_STMT | LOOP_STMT
            | IF_STMT
      STMTS = STMT | NULL

// declarations (simple, initialization, constant, and arrays)
```

```
      DECLARATION = SIMPLE_DEC | INIT_DEC | CONST_DEC | ARRAY_DEC

      SIMPLE_DEC = VAR_ID COLON DATA_TYPE SCOLON STMTS
      INIT_DEC = VAR_ID COLON DATA_TYPE EQUALS LITERAL SCOLON STMTS
      CONST_DEC = LET VAR_ID COLON DATA_TYPE EQUALS LITERAL SCOLON STMTS
      ARRAY_DEC = VAR_ID COLON DATA_TYPE ARRAY RANGE L_BRAC INT_LIT R_BRAC
                  SCOLON STMTS

      SCOLON = ';'
      EQUALS = '='
      L_BRAC = '['
      R_BRAC = ']'
      LET = 'let'
      ARRAY = 'array'
      OPT_RANGE = RANGE | NULL
      RANGE = 'range'

// literals
      LITERAL = INT_LIT | STR_LIT | CHAR_LIT | REAL_LIT | BOOL_LIT

      BOOL_LIT = TRUE | FALSE
      TRUE = 'true'
      FALSE = 'false'

// input statement
      INPUT_STMT = IN L_PAREN VAR_ID R_PAREN SCOLON STMTS
      IN = 'in'

// output statement
      OUTPUT_STMT = OUT L_PAREN OUT_ARG R_PAREN SCOLON STMTS
      OUT_ARG = STR_LIT OUT_ARGS
                  | VAR_ID OUT_ARGS
                  | NULL
      OUT_ARGS = COMMA VAR_ID OUT_ARG
                  | NULL
      OUT = 'out'

// assignment statement
      ASSIGN_STMT = VAR_ID EQUALS (LITERAL | FN_CALL | VAR_ID | EXPR) SCOLON
            STMTS

// looping statements
      LOOP_STMT = FOR_LOOP | WHILE_LOOP | DO_LOOP

// for loop
      FOR_LOOP = FOR L_PAREN VAR_ID EQUALS FOR_ARG INC_DEC FOR_ARG R_PAREN
                  L_BRC STMT R_BRC STMTS
      FOR = 'for'
      INC_DEC = TO | DOWNTO
      TO = 'to'
```

```
      DOWNTO = ‘downto’
      FOR_ARG = INT_LIT | REAL_LIT | VAR_ID

// while loop
      WHILE_LOOP = WHILE L_PAREN BOOL_EXPR R_PAREN L_BRC STMT R_BRC STMTS
      WHILE = ‘while’

// do loop
      DO_LOOP = DO L_BRC STMT R_BRC WHILE L_PAREN BOOL_EXPR R_PAREN S_COLON
                  STMTS
      DO = ‘do’

// if condition
      IF_STMT = IF_PART ELSIF_PART ELSIF_PART2 ELSE_PART STMTS

      IF_PART = IF L_PAREN BOOL_EXPR R_PAREN L_BRC STMT R_BRC

      ELSIF_PART = ELSIF L_PAREN BOOL_EXPR R_PAREN L_BRC STMT R_BRC
                  ELSIF_PART2| EMPTY
      ELSIF_PART2 = ELSIF_PART | EMPTY

      ELSE_PART = ELSE L_BRC STMT R_BRC | EMPTY

      IF = ‘if’
      ELSIF = ‘elsif’
      ELSE= ‘else’

// expressions (boolean and arithmetic)
      EXPR -> VAR_ID EXPR_PART1 | INT_LIT EXPR_PART1 | REAL_LIT EXPR_PART1
            | L_PAREN EXPR_PART3 | NOT EXPR_PART2

      EXPR_PART3 -> EXPR R_PAREN

      EXPR_PART2 -> VAR_ID | INT_LIT | REAL_LIT | L_PAREN EXPR_PART3

      EXPR_PART1 -> GRTR_THAN EXPR | LESS_THAN EXPR | EQUAL_TO EXPR
                  | NOTEQUAL EXPR | GRTR_THAN_OR_EQ EXPR
                  | LESS_THAN_OR_EQ EXPR
                  |
      EXPR_PART1 -> AND EXPR
      EXPR_PART1 -> OR EXPR
      EXPR_PART1 -> AR_OP EXPR
      EXPR_PART1 -> ''
      AR_OP -> PLUS
      AR_OP -> MINUS
      AR_OP -> MULTI
      AR_OP -> INT_DIV
      AR_OP -> REAL_DIV
      AR_OP -> MOD
      AR_OP -> EXP
```

GRAMMAR 2: LEFT-FACTORED GRAMMAR SPECIFICATIONS

CFG B = (V, T, S, P) where:

Variables (V):

```
V = { START, VIPER, HEAD, MAIN, IS, END, TAIL,
      STMT, STMTS, STMT2, STMT3, STMT4, STMT5, OUT, OUT_ARG, OUT_ARGS, IN,
      VAR_ID, FUNCTION, FUNCTIONS, FN_ID, FN_ARG, PARAM, PARAMS, FN_CALL,
      RET, LET, ARRAY, OPT_RANGE, RANGE, FOR, FOR_ARG, INC_DEC, TO, DOWNTO,
      WHILE, DO, IF, ELSE_PART, ELSE, ELSIF, ELSIF_PART, EXPR, EXPR_PART1,
      EXPR_PART2, EXPR_PART3, DATA_TYPE, INT, REAL, CHAR, CHARS, BOOL, VOID,
      LITERAL, INT_LIT, STR_LIT, CHAR_LIT, REAL_LIT, BOOL_LIT, COLON, SCOLON,
      EQUALS, L_PAREN, R_PAREN, L_BRC, R_BRC, COMMA, L_BRAC, R_BRAC,
      PLUSPLUS, SUBTSUBT, GRTR_THAN, LESS_THAN, EQUAL_TO, NOTEQUAL,
      GRTR_THAN_OR_EQ, LESS_THAN_OR_EQ, AND, OR,
      PLUS, MINUS, MULTI, INT_DIV, REAL_DIV, MOD, EXP }
```

Terminals (T)

```
T = { 'viper', 'head', 'main', 'is', 'end', 'tail',
      'let', 'array', 'range', 'returns', 'in', 'out', 'for', 'to', 'downto',
      'for', 'while', 'do', 'if', 'elsif', 'else', '&&', '||', '!', '==',
      '!=' '>', '<', '>=', '>=', '+', '-', '*', '/' '//','%', '^', '++', '--'
      '[', ']', '(', ')', ':', ',', ';', '.', '"', ''', '{', '}', '=',
      'integer', 'char', 'chars', 'real', 'boolean', 'void', 'EMPTY', 'true',
      'false'}
```

Start Symbol (S)

```
S = START
```

Productions (P):

```
// start production
      START → VIPER HEAD MAIN IS STMT END FUNCTION TAIL

// statements
     STMT → FN_ID L_PAREN FN_ARG R_PAREN SCOLON STMTS
          | IN L_PAREN VAR_ID R_PAREN SCOLON STMTS
          | OUT L_PAREN OUT_ARG R_PAREN SCOLON STMTS
          | LET VAR_ID COLON DATA_TYPE EQUALS LITERAL SCOLON STMTS
          | VAR_ID STMT2
          | FOR L_PAREN VAR_ID EQUALS FOR_ARG INC_DEC FOR_ARG R_PAREN L_BRC
         STMT R_BRC STMTS
          | WHILE L_PAREN EXPR R_PAREN L_BRC STMT R_BRC STMTS
          | DO L_BRC STMT R_BRC WHILE L_PAREN EXPR R_PAREN SCOLON STMTS
          | IF L_PAREN EXPR R_PAREN L_BRC STMT R_BRC ELSE_PART STMTS
```

```
          | PLUSPLUS VAR_ID SCOLON STMTS
          | SUBTSUBT VAR_ID SCOLON STMTS

// functions
     FUNCTION → FN_ID L_PAREN PARAM R_PAREN RET FN_RET IS STMT END FUNCTIONS
          | NULL
     FUNCTIONS → FN_ID L_PAREN PARAM R_PAREN RET FN_RET IS STMT END FUNCTION
          | NULL

// function return types
     FN_RET → DATA_TYPE
          | VOID

// function parameters
     PARAM → VAR_ID COLON DATA_TYPE PARAMS
          | NULL
     PARAMS → COMMA PARAM
          | NULL

// for loop arguments
     FOR_ARG → INT_LIT
          | REAL_LIT
          | VAR_ID

// sub-statements
     STMTS → STMT
          | NULL
     STMT2 → COLON DATA_TYPE STMT3
          | EQUALS STMT4
          | PLUSPLUS SCOLON STMTS
          | SUBTSUBT SCOLON STMTS
     STMT3 → SCOLON STMTS
          | EQUALS LITERAL SCOLON STMTS
          | ARRAY OPT_RANGE L_BRAC INT_LIT R_BRAC SCOLON STMTS
     STMT4 → LITERAL SCOLON STMTS
          | FN_CALL STMTS
          | EMP SCOLON STMTS
          | VAR_ID STMT5
     STMT5 → SCOLON STMTS
           | GRTR_THAN EXPR SCOLON STMTS
           | LESS_THAN EXPR SCOLON STMTS
           | EQUAL_TO EXPR SCOLON STMTS
```

```
           | NOTEQUAL EXPR SCOLON STMTS
           | GRTR_THAN_OR_EQ EXPR SCOLON STMTS
           | LESS_THAN_OR_EQ EXPR SCOLON STMTS
           | AND EXPR SCOLON STMTS
           | OR EXPR SCOLON STMTS
           | AR_OP EXPR SCOLON STMTS

// data types
     DATA_TYPE → INT
            | CHAR
            | CHARS
            | REAL
            | BOOL

// literals
     LITERAL → INT_LIT
          | STR_LIT
          | CHAR_LIT
          | REAL_LIT
          | BOOL_LIT

// elsif and else block
     ELSE_PART → ELSE L_BRC STMT R_BRC
          | ELSIF L_PAREN EXPR R_PAREN L_BRC STMT R_BRC ELSIF_PART
          | NULL
     ELSIF_PART → ELSE_PART
          | NULL
     OPT_RANGE → RANGE
          | NULL
     INC_DEC → TO
          | DOWNTO
     OUT_ARG → STR_LIT OUT_ARGS
          | VAR_ID OUT_ARGS
          | NULL
     OUT_ARGS → COMMA VAR_ID OUT_ARG
          | NULL

// function call
     FN_CALL → FN_ID L_PAREN FN_ARG R_PAREN SCOLON
     FN_ARG → VAR_ID FN_ARGS
          | NULL
     FN_ARGS → COMMA FN_ARG
```

```
          | NULL

// statements
     EXPR → VAR_ID EXPR_PART1
          | INT_LIT EXPR_PART1
          | REAL_LIT EXPR_PART1
          | L_PAREN EXPR_PART3
          | NOT EXPR_PART2

     EXPR_PART3 → EXPR R_PAREN

     EXPR_PART2 → VAR_ID
           | INT_LIT
           | REAL_LIT
           | L_PAREN EXPR_PART3

     EXPR_PART1 → GRTR_THAN EXPR
           | LESS_THAN EXPR
           | EQUAL_TO EXPR
           | NOTEQUAL EXPR
           | GRTR_THAN_OR_EQ EXPR
           | LESS_THAN_OR_EQ EXPR
           | AND EXPR
           | OR EXPR
           | AR_OP EXPR
           | NULL

     AR_OP → PLUS
           | MINUS
           | MULTI
           | INT_DIV
           | REAL_DIV
           | MOD
           | EXP
```

DERIVATIONS (USING LEFT-FACTORED GRAMMAR)

I. Declaration Statement

a. @var1 : integer;

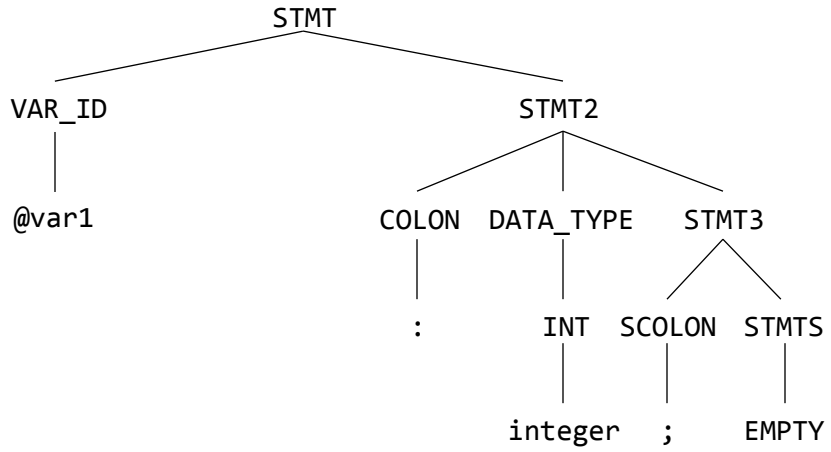
- LEFTMOST

```
STMT = VAR_ID STMT2
   = @var1 STMT2
   = @var1 COLON DATA_TYPE STMT3
   = @var1 : DATA_TYPE STMT3
   = @var1 : INT STMT3
   = @var1 : integer STMT3
   = @var1 : integer SCOLON STMTS
   = @var1 : integer SCOLON EMPTY
   = @var1 : integer SCOLON
   = @var1 : integer ;
```

- RIGHTMOST

```
STMT = VAR_ID STMT2
   = VAR_ID COLON DATA_TYPE STMT3
   = VAR_ID COLON DATA_TYPE SCOLON STMTS
   = VAR_ID COLON DATA_TYPE SCOLON EMPTY
   = VAR_ID COLON DATA_TYPE SCOLON
   = VAR_ID COLON DATA_TYPE ;
   = VAR_ID COLON INT ;
   = VAR_ID COLON integer ;
   = VAR_ID : integer ;
   = @var1 : integer ;
```

- PARSE TREE



b. @var2 : boolean = false;

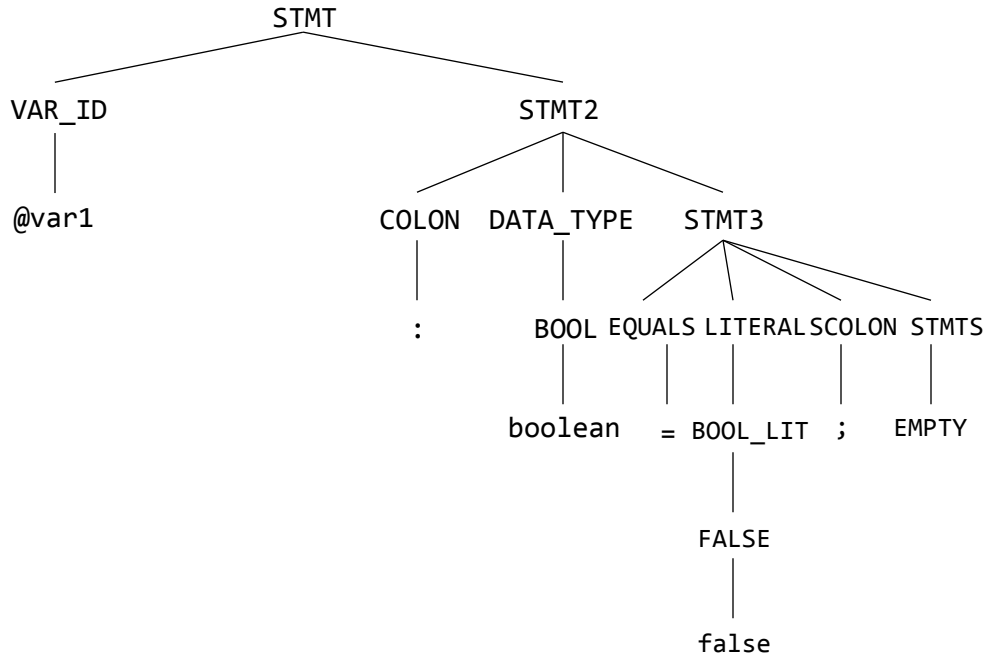
- LEFTMOST

```
STMT = VAR_ID STMT2
   = @var2 STMT2
   = @var2 COLON DATA_TYPE STMT3
   = @var2 : DATA_TYPE STMT3
   = @var2 : BOOL STMT3
   = @var2 : boolean STMT3
   = @var2 : boolean EQUALS LITERAL SCOLON STMTS
   = @var2 : boolean = LITERAL SCOLON STMTS
   = @var2 : boolean = BOOL_LIT SCOLON STMTS
   = @var2 : boolean = FALSE SCOLON STMTS
   = @var2 : boolean = false SCOLON STMTS
   = @var2 : boolean = false ; STMTS
   = @var2 : boolean = false ; EMPTY
   = @var2 : boolean = false ;
```

- RIGHTMOST

```
STMT = VAR_ID STMT2
   = VAR_ID COLON DATA_TYPE STMT3
   = VAR_ID COLON DATA_TYPE EQUALS LITERAL SCOLON STMTS
   = VAR_ID COLON DATA_TYPE EQUALS LITERAL SCOLON EMPTY
   = VAR_ID COLON DATA_TYPE EQUALS LITERAL SCOLON
   = VAR_ID COLON DATA_TYPE EQUALS LITERAL ;
   = VAR_ID COLON DATA_TYPE EQUALS BOOL_LIT ;
   = VAR_ID COLON DATA_TYPE EQUALS FALSE ;
   = VAR_ID COLON DATA_TYPE EQUALS false ;
   = VAR_ID COLON DATA_TYPE = false ;
   = VAR_ID COLON BOOL = false ;
   = VAR_ID COLON boolean = false ;
   = VAR_ID : boolean = false ;
   = @var2 : boolean = false ;
```

- PARSE TREE



II. Input Statement

a. in (@var1);

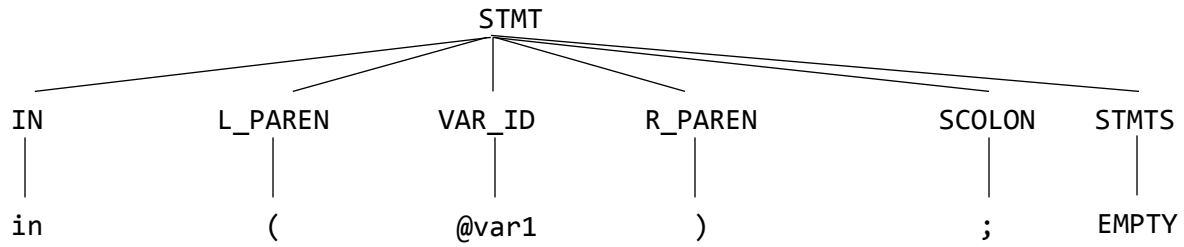
- LEFTMOST

```
STMT = IN L_PAREN VAR_ID R_PAREN SCOLON STMTS
   = in L_PAREN VAR_ID R_PAREN SCOLON STMTS
   = in ( VAR_ID R_PAREN SCOLON STMTS
   = in ( @var1 R_PAREN SCOLON STMTS
   = in ( @var1 ) SCOLON STMTS
   = in ( @var1 ) ; STMTS
   = in ( @var1 ) ; EMPTY
   = in ( @var1 ) ;
```

- RIGHTMOST

```
STMT = IN L_PAREN VAR_ID R_PAREN SCOLON STMTS
   = IN L_PAREN VAR_ID R_PAREN SCOLON EMPTY
   = IN L_PAREN VAR_ID R_PAREN SCOLON
   = IN L_PAREN VAR_ID R_PAREN ;
   = IN L_PAREN VAR_ID ) ;
   = IN L_PAREN @var1 ) ;
   = IN ( @var1 ) ;
   = in ( @var1 ) ;
```

- PARSE TREE



III. Output Statement

a. out (“Enter a number: ”);

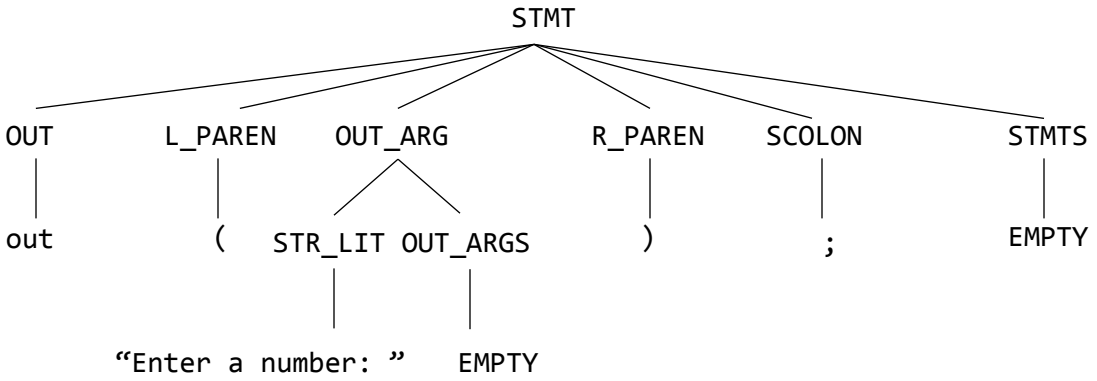
- LEFTMOST

```
OUT_STMT = OUT L_PAREN OUT_ARG R_PAREN SCOLON STMTS
   = out L_PAREN OUT_ARG R_PAREN SCOLON STMTS
   = out ( OUT_ARG R_PAREN SCOLON STMTS
   = out ( STR_LIT OUT_ARGS R_PAREN SCOLON STMTS
   = out ( “Enter a number: ” OUT_ARGS R_PAREN SCOLON STMTS
   = out ( “Enter a number: ” EMPTY R_PAREN SCOLON STMTS
   = out ( “Enter a number: ” R_PAREN SCOLON STMTS
   = out ( “Enter a number: ” ) SCOLON STMTS
   = out ( “Enter a number: ” ) ; EMPTY
   = out ( “Enter a number: ” ) ;
```

- RIGHTMOST

```
OUT_STMT = OUT L_PAREN OUT_ARG R_PAREN SCOLON STMTS
   = OUT L_PAREN OUT_ARG R_PAREN SCOLON EMPTY
   = OUT L_PAREN OUT_ARG R_PAREN SCOLON
   = OUT L_PAREN OUT_ARG R_PAREN ;
   = OUT L_PAREN OUT_ARG ) ;
   = OUT L_PAREN STR_LIT OUT_ARGS ) ;
   = OUT L_PAREN STR_LIT EMPTY ) ;
   = OUT L_PAREN STR_LIT ) ;
   = OUT L_PAREN “Enter a number: ” ) ;
   = OUT ( “Enter a number: ” ) ;
   = out ( “Enter a number: ” ) ;
```

- PARSE TREE



b. out (“Your grade is” , @grade);

- LEFTMOST

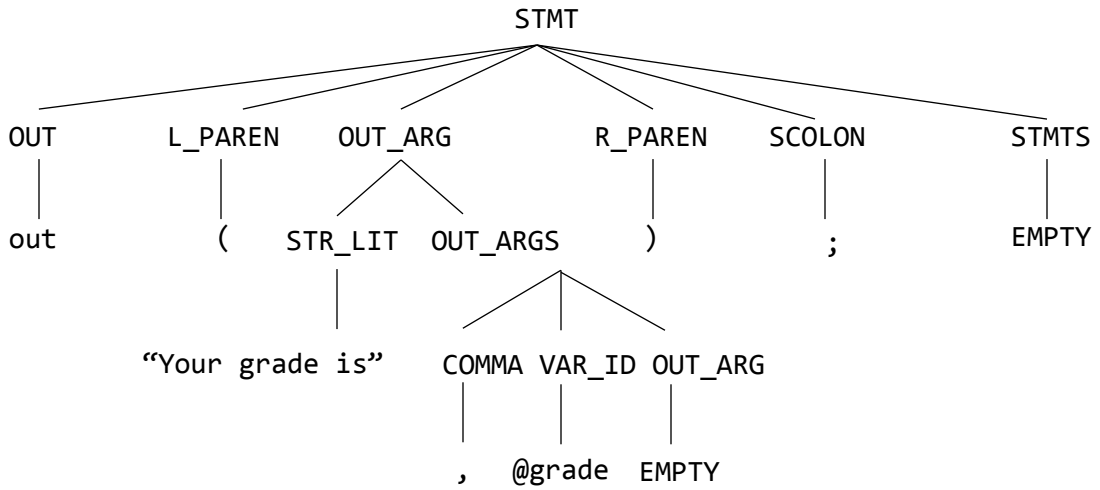
```
STMT = OUT L_PAREN OUT_ARG R_PAREN SCOLON STMTS
   = out L_PAREN OUT_ARG R_PAREN SCOLON STMTS
   = out ( OUT_ARG R_PAREN SCOLON STMTS
   = out ( STR_LIT OUT_ARGS R_PAREN SCOLON STMTS
   = out ( “Your grade is” OUT_ARGS R_PAREN SCOLON STMTS
   = out ( “Your grade is” COMMA VAR_ID OUT_ARG R_PAREN SCOLON STMTS
   = out ( “Your grade is” , VAR_ID OUT_ARG R_PAREN SCOLON STMTS
   = out ( “Your grade is” , @grade OUT_ARG R_PAREN SCOLON STMTS
   = out ( “Your grade is” , @grade EMPTY R_PAREN SCOLON STMTS
   = out ( “Your grade is” , @grade R_PAREN SCOLON STMTS
   = out ( “Your grade is” , @grade ) SCOLON STMTS
   = out ( “Your grade is” , @grade ) ; STMTS
   = out ( “Your grade is” , @grade ) ; EMPTY
   = out ( “Your grade is” , @grade ) ;
```

- RIGHTMOST

```
STMT = OUT L_PAREN OUT_ARG R_PAREN SCOLON STMTS
   = OUT L_PAREN OUT_ARG R_PAREN SCOLON EMPTY
   = OUT L_PAREN OUT_ARG R_PAREN SCOLON
   = OUT L_PAREN OUT_ARG R_PAREN ;
   = OUT L_PAREN OUT_ARG ) ;
   = OUT L_PAREN STR_LIT OUT_ARGS ) ;
   = OUT L_PAREN STR_LIT COMMA VAR_ID OUT_ARG ) ;
   = OUT L_PAREN STR_LIT COMMA VAR_ID EMPTY ) ;
   = OUT L_PAREN STR_LIT COMMA VAR_ID ) ;
   = OUT L_PAREN STR_LIT COMMA @grade ) ;
   = OUT L_PAREN STR_LIT , @grade ) ;
   = OUT L_PAREN “Your grade is” , @grade ) ;
```

```
= OUT ( “Your grade is” , @grade ) ;
= out ( “Your grade is” , @grade ) ;
```

- PARSE TREE



IV. Assignment Statement

a. @var1 = 97;

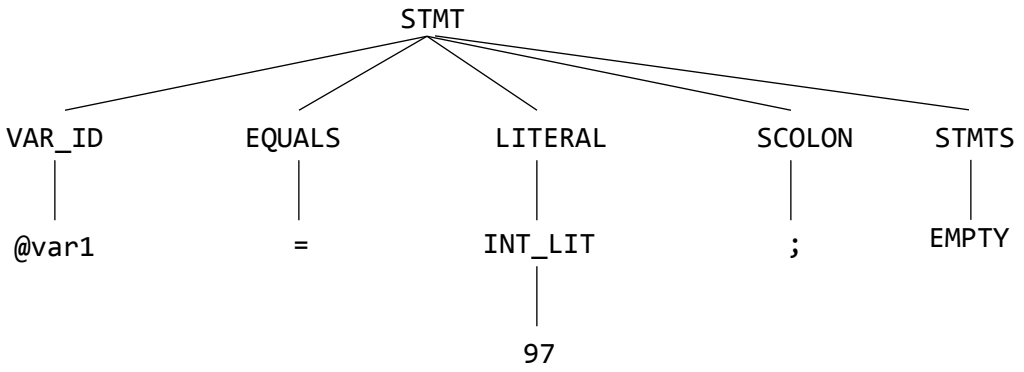
- LEFTMOST

```
STMT = VAR_ID EQUALS LITERAL SCOLON STMTS
     = @var1 EQUALS LITERAL SCOLON STMTS
     = @var1 = LITERAL SCOLON STMTS
     = @var1 = INT_LIT SCOLON STMTS
     = @var1 = 97 SCOLON STMTS
     = @var1 = 97 ; STMTS
     = @var1 = 97 ; EMPTY
     = @var1 = 97 ;
```

- RIGHTMOST

```
STMT = VAR_ID EQUALS LITERAL SCOLON STMTS
     = VAR_ID EQUALS LITERAL SCOLON EMPTY
     = VAR_ID EQUALS LITERAL SCOLON
     = VAR_ID EQUALS LITERAL ;
     = VAR_ID EQUALS INT_LIT ;
     = VAR_ID EQUALS 97 ;
     = VAR_ID = 97 ;
     = @var1 = 97 ;
```

- PARSE TREE



V. Looping Statement

a. for (@count = 1 to 6) { STMT }

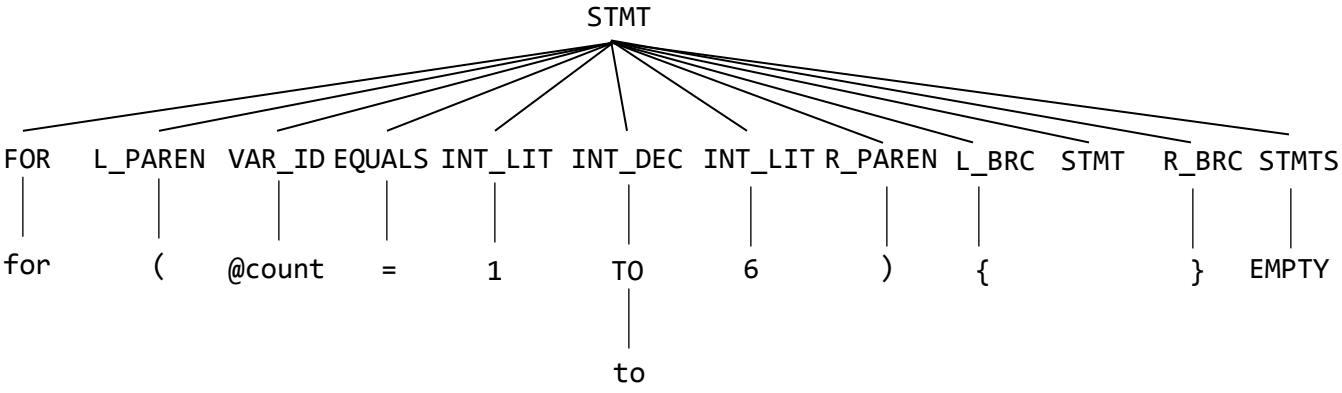
- LEFTMOST

STMT = FOR L_PAREN VAR_ID EQUALS INT_LIT INC_DEC INT_LIT
R_PAREN L_BRC
STMT R_BRC STMTS
= for L_PAREN VAR_ID EQUALS INT_LIT INC_DEC INT_LIT R_PAREN
L_BRC
STMT R_BRC STMTS
= for (VAR_ID EQUALS INT_LIT INC_DEC INT_LIT R_PAREN L_BRC
STMT R_BRC STMTS
= for (VAR_ID EQUALS INT_LIT INC_DEC INT_LIT R_PAREN L_BRC
STMT R_BRC STMTS
= for (@count EQUALS INT_LIT INC_DEC INT_LIT R_PAREN L_BRC
STMT R_BRC STMTS
= for (@count = INT_LIT INC_DEC INT_LIT R_PAREN L_BRC STMT
R_BRC STMTS
= for (@count = 1 INC_DEC INT_LIT R_PAREN L_BRC STMT R_BRC
STMTS
= for (@count = 1 TO INT_LIT R_PAREN L_BRC STMT R_BRC
STMTS
= for (@count = 1 to INT_LIT R_PAREN L_BRC STMT R_BRC
STMTS
= for (@count = 1 to 6 R_PAREN L_BRC STMT R_BRC STMTS
= for (@count = 1 to 6) L_BRC STMT R_BRC STMTS
= for (@count = 1 to 6) { STMT R_BRC STMTS
= for (@count = 1 to 6) { STMT } EMPTY
= for (@count = 1 to 6) { STMT }

- RIGHTMOST

STMT = FOR L_PAREN VAR_ID EQUALS INT_LIT INC_DEC INT_LIT
R_PAREN L_BRC
STMT R_BRC STMTS
= FOR L_PAREN VAR_ID EQUALS INT_LIT INC_DEC INT_LIT R_PAREN
L_BRC
STMT R_BRC EMPTY
= FOR L_PAREN VAR_ID EQUALS INT_LIT INC_DEC INT_LIT R_PAREN
L_BRC
STMT R_BRC
= FOR L_PAREN VAR_ID EQUALS INT_LIT INC_DEC INT_LIT R_PAREN
L_BRC
STMT }
= FOR L_PAREN VAR_ID EQUALS INT_LIT INC_DEC INT_LIT R_PAREN
{ STMT }
= FOR L_PAREN VAR_ID EQUALS INT_LIT INC_DEC INT_LIT) {
STMT }
= FOR L_PAREN VAR_ID EQUALS INT_LIT INC_DEC 6) { STMT }
= FOR L_PAREN VAR_ID EQUALS INT_LIT TO 6) { STMT }
= FOR L_PAREN VAR_ID EQUALS INT_LIT to 6) { STMT }
= FOR L_PAREN VAR_ID EQUALS 1 to 6) { STMT }
= FOR L_PAREN VAR_ID = 1 to 6) { STMT }
= FOR L_PAREN @count = 1 to 6) { STMT }
= FOR (@count = 1 to 6) { STMT }
= for (@count = 1 to 6) { STMT }

- PARSE TREE



b. while(@counter != 0){
STMT
}

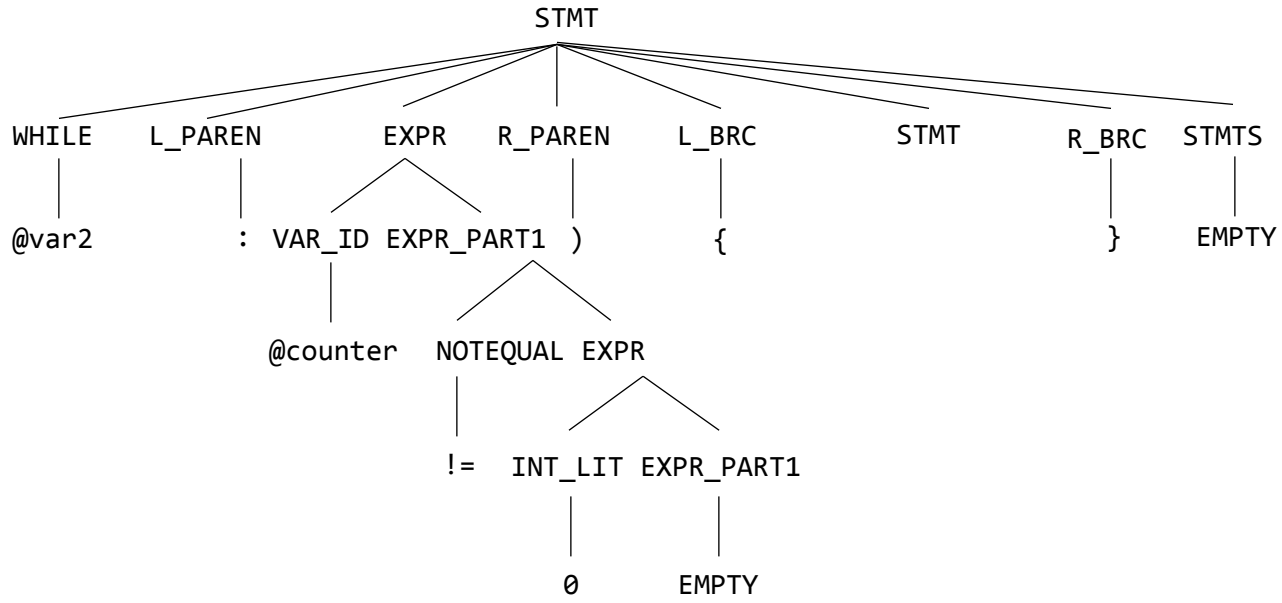
- LEFTMOST

STMT = WHILE L_PAREN EXPR R_PAREN L_BRC STMT R_BRC STMTS
= while L_PAREN EXPR R_PAREN L_BRC STMT R_BRC STMTS
= while (EXPR R_PAREN L_BRC STMT R_BRC STMTS
= while (VAR_ID EXPR_PART1 R_PAREN L_BRC STMT R_BRC STMTS
= while (@counter EXPR_PART1 R_PAREN L_BRC STMT R_BRC STMTS
= while (@counter NOTEQUAL EXPR R_PAREN L_BRC STMT R_BRC STMTS
= while (@counter != EXPR R_PAREN L_BRC STMT R_BRC STMTS
= while (@counter != INT_LIT EXPR_PART1 R_PAREN L_BRC STMT R_BRC STMTS
= while (@counter != 0 EXPR_PART1 R_PAREN L_BRC STMT R_BRC STMTS
= while (@counter != 0 EMPTY R_PAREN L_BRC STMT R_BRC STMTS
= while (@counter != 0 R_PAREN L_BRC STMT R_BRC STMTS
= while (@counter != 0) L_BRC STMT R_BRC STMTS
= while (@counter != 0) { STMT R_BRC STMTS
= while (@counter != 0) { STMT } STMTS
= while (@counter != 0) { STMT } EMPTY
= while (@counter != 0) { STMT }

- RIGHTMOST

STMT = WHILE L_PAREN EXPR R_PAREN L_BRC STMT R_BRC STMTS
= WHILE L_PAREN EXPR R_PAREN L_BRC STMT R_BRC EMPTY
= WHILE L_PAREN EXPR R_PAREN L_BRC STMT RBRC
= WHILE L_PAREN EXPR R_PAREN { STMT }
= WHILE L_PAREN EXPR) { STMT }
= WHILE L_PAREN VAR_ID EXPR_PART1) { STMT }
= WHILE L_PAREN VAR_ID NOTEQUAL EXPR) { STMT }
= WHILE L_PAREN VAR_ID NOTEQUAL INT_LIT EXPR_PART1) { STMT }
= WHILE L_PAREN VAR_ID NOTEQUAL INT_LIT EMPTY) { STMT }
= WHILE L_PAREN VAR_ID NOTEQUAL INT_LIT) { STMT }
= WHILE L_PAREN VAR_ID NOTEQUAL 0) { STMT }
= WHILE L_PAREN VAR_ID != 0) { STMT }
= WHILE L_PAREN @counter != 0) { STMT }
= WHILE (@counter != 0) { STMT }
= while (@counter != 0) { STMT }

- PARSE TREE



VI. Condition Statement

a. if(@variable == 5){
STMT
}
elsif(@variable != 5){
STMT
}

- LEFTMOST

STMT = IF L_PAREN EXPR R_PAREN L_BRC STMT R_BRC ELSE_PART
STMTS
= if L_PAREN EXPR R_PAREN L_BRC STMT R_BRC ELSE_PART STMTS
= if (EXPR R_PAREN L_BRC STMT R_BRC ELSE_PART STMTS
= if (VAR_ID EXPR_PART1 R_PAREN L_BRC STMT R_BRC ELSE_PART
STMTS
= if (@variable EXPR_PART1 R_PAREN L_BRC STMT R_BRC
ELSE_PART STMTS
= if (@variable EQUAL_TO EXPR R_PAREN L_BRC STMT R_BRC
ELSE_PART STMTS
= if (@variable == EXPR R_PAREN L_BRC STMT R_BRC ELSE_PART
STMTS
= if (@variable == INT_LIT EXPR_PART1 R_PAREN L_BRC STMT
R_BRC ELSE_PART STMTS

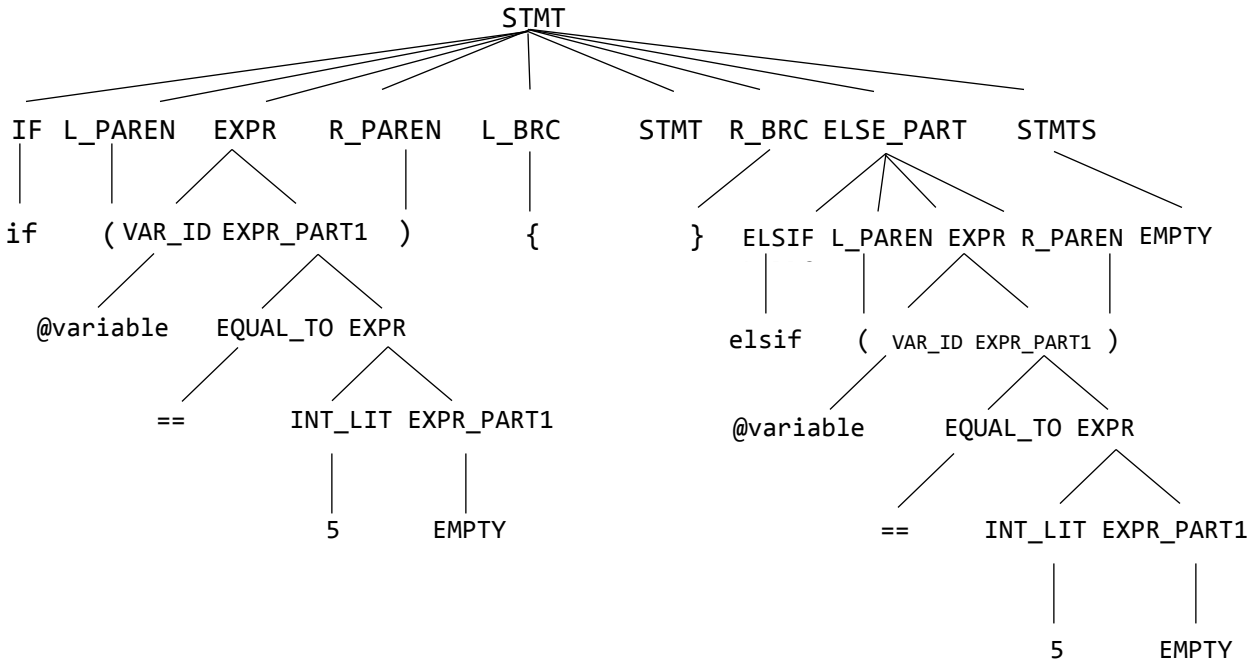
```
= if ( @variable == 5 EXPR_PART1 R_PAREN L_BRC STMT R_BRC
ELSE_PART STMTS
= if ( @variable == 5 EMPTY R_PAREN L_BRC STMT R_BRC
ELSE_PART STMTS
= if ( @variable == 5 R_PAREN L_BRC STMT R_BRC ELSE_PART
STMTS
= if ( @variable == 5 ) L_BRC STMT R_BRC ELSE_PART STMTS
= if ( @variable == 5 ) { STMT R_BRC ELSE_PART STMTS
= if ( @variable == 5 ) { STMT } ELSE_PART STMTS
= if ( @variable == 5 ) { STMT } ELSIF L_PAREN EXPR R_PAREN
L_BRC STMT R_BRC ELSIF_PART STMTS
= if ( @variable == 5 ) { STMT } elsif L_PAREN EXPR R_PAREN
L_BRC STMT R_BRC ELSIF_PART STMTS
= if ( @variable == 5 ) { STMT } elsif ( EXPR R_PAREN L_BRC
STMT R_BRC ELSIF_PART STMTS
= if ( @variable == 5 ) { STMT } elsif ( VAR_ID EXPR_PART1
R_PAREN L_BRC STMT R_BRC ELSIF_PART STMTS
= if ( @variable == 5 ) { STMT } elsif ( @variable
EXPR_PART1 R_PAREN L_BRC STMT R_BRC ELSIF_PART STMTS
= if ( @variable == 5 ) { STMT } elsif ( @variable NOTEQUAL
EXPR R_PAREN L_BRC STMT R_BRC ELSIF_PART STMTS
= if ( @variable == 5 ) { STMT } elsif ( @variable != EXPR
R_PAREN L_BRC STMT R_BRC ELSIF_PART STMTS
= if ( @variable == 5 ) { STMT } elsif ( @variable !=
INT_LIT EXPR_PART1 R_PAREN L_BRC STMT R_BRC ELSIF_PART
STMTS
= if ( @variable == 5 ) { STMT } elsif ( @variable != 5
EXPR_PART1 R_PAREN L_BRC STMT R_BRC ELSIF_PART STMTS
= if ( @variable == 5 ) { STMT } elsif ( @variable != 5
EMPTY R_PAREN L_BRC STMT R_BRC ELSIF_PART STMTS
= if ( @variable == 5 ) { STMT } elsif ( @variable != 5
R_PAREN L_BRC STMT R_BRC ELSIF_PART STMTS
= if ( @variable == 5 ) { STMT } elsif ( @variable != 5 )
L_BRC STMT R_BRC ELSIF_PART STMTS
= if ( @variable == 5 ) { STMT } elsif ( @variable != 5 ) {
STMT R_BRC ELSIF_PART STMTS
= if ( @variable == 5 ) { STMT } elsif ( @variable != 5 ) {
STMT } ELSIF_PART STMTS
= if ( @variable == 5 ) { STMT } elsif ( @variable != 5 ) {
STMT } EMPTY STMTS
= if ( @variable == 5 ) { STMT } elsif ( @variable != 5 ) {
STMT } STMTS
= if ( @variable == 5 ) { STMT } elsif ( @variable != 5 ) {
STMT } EMPTY
= if ( @variable == 5 ) { STMT } elsif ( @variable != 5 ) {
STMT }
```

- RIGHTMOST

```
STMT = IF L_PAREN EXPR R_PAREN L_BRC STMT R_BRC ELSE_PART
   STMTS
   = IF L_PAREN EXPR R_PAREN L_BRC STMT R_BRC ELSE_PART EMPTY
   = IF L_PAREN EXPR R_PAREN L_BRC STMT R_BRC ELSE_PART
   = IF L_PAREN EXPR R_PAREN L_BRC STMT R_BRC ELSIF L_PAREN
EXPR R_PAREN L_BRC STMT R_BRC ELSIF_PART
   = IF L_PAREN EXPR R_PAREN L_BRC STMT R_BRC ELSIF L_PAREN
EXPR R_PAREN L_BRC STMT R_BRC EMPTY
   = IF L_PAREN EXPR R_PAREN L_BRC STMT R_BRC ELSIF L_PAREN
EXPR R_PAREN L_BRC STMT R_BRC
   = IF L_PAREN EXPR R_PAREN L_BRC STMT R_BRC ELSIF L_PAREN
EXPR R_PAREN L_BRC STMT }
   = IF L_PAREN EXPR R_PAREN L_BRC STMT R_BRC ELSIF L_PAREN
EXPR R_PAREN { STMT }
   = IF L_PAREN EXPR R_PAREN L_BRC STMT R_BRC ELSIF L_PAREN
EXPR ) { STMT }
   = IF L_PAREN EXPR R_PAREN L_BRC STMT R_BRC ELSIF L_PAREN
VAR_ID EXPR_PART1 ) { STMT }
   = IF L_PAREN EXPR R_PAREN L_BRC STMT R_BRC ELSIF L_PAREN
VAR_ID NOTEQUAL EXPR ) { STMT }
   = IF L_PAREN EXPR R_PAREN L_BRC STMT R_BRC ELSIF L_PAREN
VAR_ID NOTEQUAL INT_LIT EXPR_PART1 ) { STMT }
   = IF L_PAREN EXPR R_PAREN L_BRC STMT R_BRC ELSIF L_PAREN
VAR_ID NOTEQUAL INT_LIT EMPTY ) { STMT }
   = IF L_PAREN EXPR R_PAREN L_BRC STMT R_BRC ELSIF L_PAREN
VAR_ID NOTEQUAL INT_LIT ) { STMT }
   = IF L_PAREN EXPR R_PAREN L_BRC STMT R_BRC ELSIF L_PAREN
VAR_ID NOTEQUAL 5 ) { STMT }
   = IF L_PAREN EXPR R_PAREN L_BRC STMT R_BRC ELSIF L_PAREN
VAR_ID != 5 ) { STMT }
   = IF L_PAREN EXPR R_PAREN L_BRC STMT R_BRC ELSIF L_PAREN
@variable != 5 ) { STMT }
   = IF L_PAREN EXPR R_PAREN L_BRC STMT R_BRC ELSIF (
@variable != 5 ) { STMT }
   = IF L_PAREN EXPR R_PAREN L_BRC STMT R_BRC elsif (
@variable != 5 ) { STMT }
   = IF L_PAREN EXPR R_PAREN L_BRC STMT } elsif ( @variable !=
5 ) { STMT }
   = IF L_PAREN EXPR R_PAREN { STMT } elsif ( @variable != 5 )
{ STMT }
   = IF L_PAREN EXPR ) { STMT } elsif ( @variable != 5 ) {
STMT }
   = IF L_PAREN VAR_ID EXPR_PART1 ) { STMT } elsif ( @variable
!= 5 ) { STMT }
   = IF L_PAREN VAR_ID EQUAL_TO EXPR ) { STMT } elsif (
@variable != 5 ) { STMT }
   = IF L_PAREN VAR_ID EQUAL_TO INT_LIT EXPR_PART1 ) { STMT }
elsif ( @variable != 5 ) { STMT }
```

```
   = IF L_PAREN VAR_ID EQUAL_TO INT_LIT EMPTY ) { STMT } elsif
( @variable != 5 ) { STMT }
   = IF L_PAREN VAR_ID EQUAL_TO INT_LIT ) { STMT } elsif (
@variable != 5 ) { STMT }
   = IF L_PAREN VAR_ID EQUAL_TO 5 ) { STMT } elsif ( @variable
!= 5 ) { STMT }
   = IF L_PAREN VAR_ID == 5 ) { STMT } elsif ( @variable != 5
) { STMT }
   = IF L_PAREN @variable == 5 ) { STMT } elsif ( @variable !=
5 ) { STMT }
   = IF ( @variable == 5 ) { STMT } elsif ( @variable != 5 ) {
STMT }
   = if ( @variable == 5 ) { STMT } elsif ( @variable != 5 ) {
STMT }
```

- PARSE TREE



ERROR RECOVERY

The parser of Viper Programming Language uses **PANIC MODE ERROR RECOVERY STRATEGY**.

It is defined in Tutorials Point as a strategy that when a parser encounters an error anywhere in the statement, it ignores the rest of the statement by not processing input from erroneous input to delimiter, such as semi-colon. This is the easiest way of error-recovery and also, it prevents the parser from developing infinite loops.

ERROR MESSAGES

Viper parser produces the following messages when encounters an error:

"Unexpected token "<token>", lexeme: "<lexeme>" on line <line number>"
"Expects a token to be: <token>"

SAMPLE PROGRAM

```
viper head
      main is
            @_base : real;
            @_limit : real;

            out("enter base number:");
            in(@_base);

            out("enter limit number:");
            in(@_limit);

            while (@_base != @_limit) {
                  @_i++;
            }

            if (@_limit == @base) {
                  out("Success");
            }
            else {
                  @_limit = 4;
            }
      end
tail
```

FIRST AND FOLLOW SETS

TOKEN	FIRST	FOLLOW
START	{VIPER}	{$}
STMT	{FN_ID,IN,OUT,LET,VAR_ID,FOR,WHILE,DO,IF,PLUSPLUS,SUBTSUBT}	{END,R_BRC}
FUNCTION	{FN_ID,''}	{TAIL}
FUNCTIONS	{FN_ID,''}	{TAIL}
FN_RET	{VOID,INT,CHAR,CHARS,REAL,BOOL}	{IS}
PARAM	{VAR_ID,''}	{R_PAREN}
PARAMS	{COMMA,''}	{R_PAREN}
FOR_ARG	{INT_LIT,REAL_LIT,VAR_ID}	{TO,DOWNTO,R_PAREN}
STMT2	{COLON,EQUALS,PLUSPLUS,SUBTSUBT}	{END,R_BRC}
STMT3	{SCOLON,EQUALS,ARRAY}	{END,R_BRC}
STMT4	{EMP,VAR_ID,INT_LIT,STR_LIT,CHAR_LIT,REAL_LIT,BOOL_LIT,FN_ID}	{END,R_BRC}
STMT5	{SCOLON,GRTR_THAN,LESS_THAN,EQUAL_TO,NOTEQUAL,GRTR_THAN_OR_EQ,LESS_THAN_OR_EQ,AND,OR,PLUS,MINUS,MULTI,INT_DIV,REAL_DIV,MOD,EXP}	{END,R_BRC}
DATA_TYPE	{INT,CHAR,CHARS,REAL,BOOL}	{EQUALS,IS,COMMA,R_PAREN,SCOLON,ARRAY}
LITERAL	{INT_LIT,STR_LIT,CHAR_LIT,REAL_LIT,BOOL_LIT}	{SCOLON}
ELSE_PART	{ELSE,ELSIF,''}	{FN_ID,IN,OUT,LET,VAR_ID,FOR,WHILE,DO,IF,PLUSPLUS,SUBTSUBT,END,R_BRC}
ELSIF_PART	{ELSE,ELSIF,''}	{FN_ID,IN,OUT,LET,VAR_ID,FOR,WHILE,DO,IF,PLUSPLUS,SUBTSUBT,END,R_BRC}
OPT_RANGE	{RANGE,''}	{L_BRAC}
INC_DEC	{TO,DOWNTO}	{INT_LIT,REAL_LIT,VAR_ID}
OUT_ARG	{STR_LIT,VAR_ID,''}	{R_PAREN}
OUT_ARGS	{COMMA,''}	{R_PAREN}
FN_CALL	{FN_ID}	{FN_ID,IN,OUT,LET,VAR_ID,FOR,WHILE,DO,IF,PLUSPLUS,SUBTSUBT,END,R_BRC}
FN_ARG	{VAR_ID,''}	{R_PAREN}
FN_ARGS	{COMMA,''}	{R_PAREN}
STMTS	{FN_ID,IN,OUT,LET,VAR_ID,FOR,WHILE,DO,IF,PLUSPLUS,SUBTSUBT,''}	{END,R_BRC}
EXPR	{VAR_ID,INT_LIT,REAL_LIT,L_PAREN,NOT}	{R_PAREN,SCOLON}
EXPR_PART3	{VAR_ID,INT_LIT,REAL_LIT,L_PAREN,NOT}	{R_PAREN,SCOLON}

EXPR_PART2	{VAR_ID,INT_LIT,REAL_LIT,L_PAREN,NOT}	{R_PAREN,SCOLON}
EXPR_PART1	{GRTR_THAN,LESS_THAN,EQUAL_TO,NOTEQUAL,GRTR_THAN_OR_EQ,LESS_THAN_OR_EQ,AND,OR,'',PLUS,MINUS,MULTI,INT_DIV,REAL_DIV,MOD,EXP}	{R_PAREN,SCOLON}
AR_OP	{PLUS,MINUS,MULTI,INT_DIV,REAL_DIV,MOD,EXP}	{VAR_ID,INT_LIT,REAL_LIT,L_PAREN,NOT}