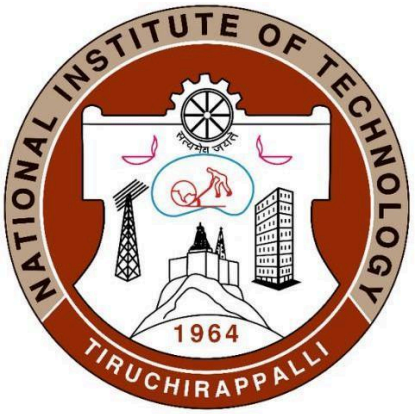
NATIONAL INSTITUTE OF TECHNOLOGY, TIRUCHIRAPPALLI



CSPC62

COMPILER DESIGN

TOPIC : Compiler for Base Typescript

LAB REPORT – 2

Sub Topic: Syntax Analyser

DONE BY

S.No	Name	Roll No.
1.	Dhrubit Hajong	106121037
2.	Mercia Melvin Perinchery	106121077
3.	Nishith Eedula	106121085

Index

Phase 2: Syntax Analyser

Phase 2: Syntax Analyser

I. Creating the Parser using YACC

A **parser** is a program used to check whether the given input is syntactically correct, i.e. in accordance with the production rules that specify the language's grammar, usually a CFG.

A **Context Free Grammar (CFG)** is defined by a four tuple (N, T, P, S) where

N is the set of non-terminals
T is the set of terminals. Here, it consists of the tokens returned by the Lexical Analyser.
P is the set of production rules
S is the start symbol.

Each production rule consists of a non-terminal on the left-hand side and a sequence of terminals (tokens) and non-terminals on the right-hand side.

YACC (Yet Another Compiler Compiler) is a Unix compiler that can generate a LALR(1) parser for other programming languages. YACC translates the given Context Free Grammar (CFG) specifications into the C implementation (*y.tab.c*) of the corresponding push-down automata of the language. When compiled, it yields an executable parser.

A YACC program typically consists of three sections: the Declarations, the Rules and the Auxiliary functions or Subroutines.

```
DECLARATIONS

%%

RULES

%%

AUXILIARY FUNCTIONS
```

Declarations

This section consists of two parts.

a. The C Declarations

Delimited by `%{` and `%}`, consisting of all the declarations required for the C code written under the Actions section and the Auxiliary functions section. The contents of this section are copied into the *y.tab.c* file without any modifications.

b. The YACC Declarations

Comprise the tokens' declarations (returned by the lexical analyser).

```
/* Definitions */
%{
    #include <stdio.h>
    #include <string.h>
    #include <stdlib.h>
    #include <ctype.h>
    #include "lex.yy.c"

    int yyerror(const char *s);
    int yylex(void);
    int yywrap();
    int success = 1;
%}


/* Declarations */
%token LET CONST VAR IF ELSE WHILE RETURN ANYTYPE NUMBERTYPE STRINGTYPE
BOOLEANTYPE UNDEFINEDTYPE TYPE FUNCTION ADD SUBTRACT MULTIPLY DIVIDE MODULO
EXPONENT TRUE FALSE AND OR EQUAL DOUBLEEQUAL NOTEQUAL LESSTHAN LESSEQUAL
GREATERTHAN GREATEREQUAL UNION INTERSECTION INTEGERVALUE FLOATVALUE
STRINGVALUE IDENTIFIER CONSOLELOG

%define parse.error verbose
```

Rules

Rules in a YACC program consist of two parts - the production and the action. A rule in YACC is of the form.

```
Production_head : production_body {action in C } ;
```

The abstract outline of the structure of the rules part of the YACC program.

```
%%
/* Rules Section begins here */

/* Rules Section ends here */

%%
```

Productions

The head (LHS) of a production is always a non-terminal. Every non-terminal used in the grammar must appear as the head of at least one production. A non-terminal in the head of the production may have one or more production bodies separated by a “|”.

On the other hand, the body (RHS) of a production consists of a sequence of terminals (tokens) and non-terminals. These terminals, or tokens, have been previously declared as a part of the Declarations.

Some of the Production Rules that we defined for Base Typescript include

1. Body

The body refers to the sequence of blocks comprising declarations and statements that make up the program. It may or may not be empty.

```
body: block body
|
;
```

2. Block

As mentioned, a block consists of a sequence of declarations and statements. Here, we have defined the *while, if, function declaration*, & *consolelog* statements as part of the block production rule, in addition to the general *statement* structure.

```
block: WHILE '(' condition ')' '{' body '}'
| IF '(' condition ')' '{' body '}' else
| FUNCTION IDENTIFIER '(' parameter ')' '{' body '}'
| statement semicolon
| CONSOLELOG '(' STRINGVALUE ')' semicolon
;
```

3. Declarations and Scope

A declaration is a statement that binds an identifier/variable to a constant, expression or function.

a. Variable Declaration

```
declaration: LET
| VAR
| CONST
;

parameter: parameter ',' parameter
| IDENTIFIER ':' datatype
|
;

datatype: NUMBERTYPE
| STRINGTYPE
| BOOLEANTYPE
;

statement: declaration IDENTIFIER ':' datatype init {$4.nd = mknode(NULL, NULL, $4.name); $2.nd = mknode(NULL, NULL, $2.name); $1.nd = mknode($4.nd, $2.nd, $1.name); $$.nd = mknode($1.nd, $5.nd, "Initialisation");}
| IDENTIFIER '=' expression { $1.nd = mknode(NULL, NULL, $1.name); $$.nd = mknode($1.nd, $3.nd, "="); }
| IDENTIFIER relop expression { $1.nd = mknode(NULL, NULL, $1.name); $$.nd = mknode($1.nd, $3.nd, $2.name ); }
;
```

b. Function Declaration

```
function: FUNCTION IDENTIFIER '(' parameter ')' '{' body return '}' { struct node *main = mknode($7.nd, $8.nd, $2.name); $$.nd = mknode($1.nd, main, "Function"); head = $$.nd; }
;
```

4. Expressions (Arithmetic, Relational and Logical)

An expression is a combination of operators, constants and variables to produce or represent a given value.

a. Arithmetic Expressions

```
expression : expression addops term { $$.nd = mknode($1.nd, $3.nd, $2.name); }
| term { $$.nd = $1.nd;}
;

term : term mulops factor { $$.nd = mknode($1.nd, $3.nd, $2.name); }
```

```
| factor {$$.nd = $1.nd;}
;

factor : base exponent base { $$.nd = mknode($1.nd, $3.nd, $2.name); }
| LOG '(' value ',' value ')' {$$.nd = mknode($3.nd, $5.nd, $1.name); }
| base {$$.nd = $1.nd;}
;

base : value {$$.nd = $1.nd;}
| '(' expression ')' {$$.nd = $2.nd;}
;

exponent: POW
;

mulops: MULT
| DIV
;

addops: ADD
| SUB
;
```

b. Relational Expressions

```
relop: LT
| GT
| LE
| GE
| EQ
| NE
;
```

c. Logical Expressions

```
block: WHILE '(' condition ')''{' body '}' { $$.nd = mknode($3.nd, $6.nd,
$1.name); }
| IF '(' condition ')''{' body '}' else { struct node *iff = mknode($3.nd,
$6.nd, $1.name); $$.nd = mknode(iff, $8.nd, "conditionalBranch"); }
;

else: ELSE '{' body '}' { struct node *cond = mknode(NULL, NULL,
"EndOfConditional"); $$.nd = mknode($3.nd, cond, $1.name); }
| { $$.nd = NULL; }
;

condition: condition and_or condition { $$.nd = mknode($1.nd, $3.nd, $2.name);
}
| value relop value { $$.nd = mknode($1.nd, $3.nd, $2.name);}
```

```
| value { $$.nd = $1.nd;}
| TRUE {$$.nd = NULL; }
| FALSE {$$.nd = NULL; }
;
```

Actions

The action part of a rule consists of those C statements enclosed with curly brackets. When executed, these statements match the input with the body of a production rule, and a reduction occurs.

Auxiliary Functions/Subroutines

This section contains the three mandatory functions:

1. **main()**
   Invokes the yyparse() function to parse the given input file. It executes the relevant action by matching the input to its appropriate production body.
2. **yylex()**
   The parser invokes this function to read the tokens. Each invocation of yylex() should return the next token (from the input stream) to yyparse().
3. **yyerror()**
   When the parser fails to find any matching body part, this function is invoked to print the error message.

As mentioned previously, YACC generates a LALR(1) parser. The LALR(1) parser is a push-down automaton consisting of a finite state machine with a stack that holds terminal and/or non-terminal symbols. The parser works by repeatedly performing the four possible parser actions:

1. **Shift** is the parser action of removing the next unread terminal from the input buffer and pushing it into the stack. (The input terminal gets “shifted” to the stack).
2. **Reduce** is the parser action of replacing one or more grammar symbols from the top of the stack that matches the body of a production with the corresponding production head. The contents on top of the stack, which matches the right side of a production, is called a handle. Replacing a handle with the corresponding production head is called a reduction.
3. **Accept** is the parser action indicating that the entire input has been parsed successfully. The parser executes an accept action only if it reaches the accepting configuration – one in which the input buffer is empty and the stack contains just the start variable followed by '$'. Accepting state would be of the form:
4. **Error** indicates that an error was encountered while parsing the input. In our example, there is no error. We will see error conditions later.

Parsing ends successfully when the input buffer is empty (except for the end-marker '$'), and the stack contains nothing but the '$' followed by the grammar's start symbol.

```c
/* Main Function */
int main() {
    extern FILE *yyin, *yyout;

    int p = -1;
    yydebug = 1;
    p = yyparse();
    if(p)
        printf("Parsing Successful\n");
    printf("\033[4mParse Tree\033[24m");
    printf("\n\n");
      printBT(head);
    printf("\n\n");


    return p;
}

/* Error Handling */
int yyerror(const char *msg)
{
    extern int yylineno;
    printf("%sParsing Failed\nLine Number: %d,%s\n",BOLDRED,yylineno,msg);
    exit(0);
    return 0;
}
```

II. YACC Code for Implementing the Base Typescript Parser and Creating the Syntax Tree

```
%{
    #include <stdio.h>
    #include <string.h>
    #include <stdlib.h>
    #include <ctype.h>
    #include "lex.yy.c"

    #define BOLDRED "\033[1m\033[31m"
    #define YYDEBUG 1

    int yyerror(const char *s);
    int yylex(void);
    int yywrap();


    struct node* mknode(struct node *left, struct node *right, char *token);
    void printBT(struct node*);

    struct node *head;
    struct node {
        struct node *left;
        struct node *right;
        char *token;
    };
%}


%union {
       struct var_name {
              char name[100];
              struct node* nd;
       } treeNode;
}

%token <treeNode> CONSOLELOG SCAN IF WHILE ELSE RETURN ELIF LET VAR CONST ADD SUB MULT
DIV LOG POW GE LE GT LT EQ NE TRUE FALSE AND OR NUMBERTYPE STRINGTYPE BOOLEANTYPE
FUNCTION INTEGER FLOAT IDENTIFIER STRINGVALUE

%type <treeNode> function entry parameter datatype body block else condition statement
declaration init expression term factor base exponent mulops addops relop number value
return and_or
%define parse.error verbose
%start body


%%

function: FUNCTION IDENTIFIER '(' parameter ')' '{' body return '}' { struct node
*main = mknode($7.nd, $8.nd, $2.name); $$.nd = mknode($1.nd, main, "Function"); head =
```

```
$$.nd; }
;

parameter: parameter ',' parameter
| IDENTIFIER ':' datatype
|
;

datatype: NUMBERTYPE
| STRINGTYPE
| BOOLEANTYPE
;

body: block body {$$.nd = mknode($1.nd, $2.nd, "Scope");}
| { $$.nd = mknode(NULL, NULL, "EndOfScope"); }
;

block: WHILE '(' condition ')''{' body '}' { $$.nd = mknode($3.nd, $6.nd, $1.name); }
| IF '(' condition ')''{' body '}' else { struct node *iff = mknode($3.nd, $6.nd,
$1.name); $$.nd = mknode(iff, $8.nd, "conditionalBranch"); }
| statement ';' { $$.nd = $1.nd; }
| CONSOLELOG '(' STRINGVALUE ')' ';' { struct node *data = mknode(NULL, NULL,
$3.name); $$.nd = mknode(NULL, data, "ConsoleLog"); }
| CONSOLELOG '(' IDENTIFIER ')' ';' { struct node *data = mknode(NULL, NULL, $3.name);
$$.nd = mknode(NULL, data, "ConsoleLog"); }
;

else: ELSE '{' body '}' {  struct node *cond = mknode(NULL, NULL, "EndOfConditional");
$$.nd = mknode($3.nd, cond, $1.name); }
| { $$.nd = NULL; }
;

condition: condition and_or condition { $$.nd = mknode($1.nd, $3.nd, $2.name); }
| value relop value { $$.nd = mknode($1.nd, $3.nd, $2.name);}
| value { $$.nd = $1.nd;}
| TRUE {$$.nd = NULL; }
| FALSE {$$.nd = NULL; }
;

statement: declaration IDENTIFIER ':' datatype init {$4.nd = mknode(NULL, NULL,
$4.name); $2.nd = mknode(NULL, NULL, $2.name); $1.nd = mknode($4.nd, $2.nd, $1.name);
$$.nd = mknode($1.nd, $5.nd, "Initialisation");}
| IDENTIFIER '=' expression { $1.nd = mknode(NULL, NULL, $1.name); $$.nd =
mknode($1.nd, $3.nd, "="); }
| IDENTIFIER relop expression { $1.nd = mknode(NULL, NULL, $1.name); $$.nd =
mknode($1.nd, $3.nd, $2.name ); }
;

declaration: LET
| VAR
| CONST
;
```

```
init: '=' value { $$.nd = $2.nd;}
| '=' expression { $$.nd = $2.nd;}
| { $$.nd = NULL; }
;


expression : expression addops term { $$.nd = mknode($1.nd, $3.nd, $2.name); }
| term { $$.nd = $1.nd;}
;

term : term mulops factor { $$.nd = mknode($1.nd, $3.nd, $2.name); }
| factor {$$.nd = $1.nd;}
;

factor : base exponent base { $$.nd = mknode($1.nd, $3.nd, $2.name); }
| LOG '(' value ',' value ')' {$$.nd = mknode($3.nd, $5.nd, $1.name); }
| base {$$.nd = $1.nd;}
;

base : value {$$.nd = $1.nd;}
| '(' expression ')' {$$.nd = $2.nd;}
;

and_or : AND { $$.nd = mknode(NULL, NULL, $1.name); }
| OR { $$.nd = mknode(NULL, NULL, $1.name); }
;

exponent: POW
;

mulops: MULT
| DIV
;

addops: ADD
| SUB
;
relop: LT
| GT
| LE
| GE
| EQ
| NE
;

number: INTEGER
| FLOAT
;

value: number {$$.nd = mknode(NULL, NULL, $1.name);}
| IDENTIFIER {$$.nd = mknode(NULL, NULL, $1.name);}
| STRINGVALUE {$$.nd = mknode(NULL, NULL, $1.name);}
| TRUE {$$.nd = mknode(NULL, NULL, $1.name);}
```

```
| FALSE {$$.nd = mknode(NULL, NULL, $1.name);}
| SCAN '('')' { $$.nd = mknode(NULL, NULL, "scan"); }
;

return: RETURN value ';'  { $1.nd = mknode(NULL, NULL, "return"); $$.nd =
mknode($1.nd, $2.nd, "ReturnStatement"); }
| { $$.nd = NULL; }
;

%%


int main() {
    extern FILE *yyin, *yyout;

    int p = -1;
    p = yyparse();
    yydebug = 1;
    if(p)
        printf("Parsing Successful\n");
    printf("\033[4mParse Tree\033[24m");
    printf("\n\n");
       printBT(head);
    printf("\n\n");


    return p;
}

int yyerror(const char *msg)
{
    extern int yylineno;
    printf("%sParsing Failed\nLine Number: %d,%s\n",BOLDRED,yylineno,msg);
    exit(0);
    return 0;
}

void printBTHelper(char* prefix, struct node* ptr, int isLeft) {
    if( ptr != NULL ) {
        printf("%s",prefix);
        if(isLeft) { printf("├──"); }
             else { printf("└──"); }
        printf("%s",ptr->token);
             printf("\n");
             char* addon = isLeft ? "│   " : "    ";
      int len2 = strlen(addon);
      int len1 = strlen(prefix);
      char* result = (char*)malloc(len1 + len2 + 1);
      strcpy(result, prefix);
      strcpy(result + len1, addon);
             printBTHelper(result, ptr->left, 1);
             printBTHelper(result, ptr->right, 0);
      free(result);
```

```
    }
}

void printBT(struct node* ptr) {
      printf("\n");
    printBTHelper("", ptr, 0);
}

struct node* mknode(struct node *left, struct node *right, char *token) {
      struct node *newnode = (struct node *)malloc(sizeof(struct node));
      char *newstr = (char *)malloc(strlen(token)+1);
      strcpy(newstr, token);
      newnode->left = left;
      newnode->right = right;
      newnode->token = newstr;
      return(newnode);
}
```

III. Sample Input Program and Terminal Output

Input Program (TypeScript File)

```
// Ignored
/* Ignored */

function trialFunction (param1: number) {
    let valueNumber:number = 3;
    const valueBoolean:boolean = true;
    var valueExpression:number = 5 * 6;

    if(valueNumber > valueExpression && valueBoolean == true){
        console.log("ABC");
    }
    else{
        console.log(valueBoolean);
    }

    let loopValue:number = 10;
    while(loopValue) {
        loopValue = loopValue - 1;
    }

    return 3;
}
```

Terminal Output: Step by Step Parsing Process

```
Starting parse

Entering state 0
Reading a token: Next token is token FUNCTION ()
Shifting token FUNCTION ()

Entering state 1
Reading a token: Next token is token IDENTIFIER ()
Shifting token IDENTIFIER ()

Entering state 3
Reading a token: Next token is token '(' ()
Shifting token '(' ()

Entering state 5
Reading a token: Next token is token IDENTIFIER ()
Shifting token IDENTIFIER ()

Entering state 6
Reading a token: Next token is token ':' ()
Shifting token ':' ()

Entering state 8
Reading a token: Next token is token NUMBERTYPE ()
Shifting token NUMBERTYPE ()

Entering state 11
Reducing stack by rule 5 (line 50):
   $1 = token NUMBERTYPE ()
-> $$ = nterm datatype ()
Stack now 0 1 3 5 6 8

Entering state 14
Reducing stack by rule 3 (line 46):
   $1 = token IDENTIFIER ()
   $2 = token ':' ()
   $3 = nterm datatype ()
-> $$ = nterm parameter ()
Stack now 0 1 3 5

Entering state 7
Reading a token: Next token is token ')' ()
Shifting token ')' ()

Entering state 9
Reading a token: Next token is token '{' ()
Shifting token '{' ()

Entering state 15
Reading a token: Next token is token LET ()
Shifting token LET ()
```

```
Entering state 20
Reducing stack by rule 25 (line 82):
   $1 = token LET ()
-> $$ = nterm declaration ()
Stack now 0 1 3 5 7 9 15

Entering state 27
Reading a token: Next token is token IDENTIFIER ()
Shifting token IDENTIFIER ()

Entering state 43
Reading a token: Next token is token ':' ()
Shifting token ':' ()

Entering state 69
Reading a token: Next token is token NUMBERTYPE ()
Shifting token NUMBERTYPE ()

Entering state 11
Reducing stack by rule 5 (line 50):
   $1 = token NUMBERTYPE ()
-> $$ = nterm datatype ()
Stack now 0 1 3 5 7 9 15 27 43 69

Entering state 90
Reading a token: Next token is token '=' ()
Shifting token '=' ()

Entering state 103
Reading a token: Next token is token INTEGER ()
Shifting token INTEGER ()

Entering state 49
Reducing stack by rule 53 (line 133):
   $1 = token INTEGER ()
-> $$ = nterm number ()
Stack now 0 1 3 5 7 9 15 27 43 69 90 103

Entering state 54
Reducing stack by rule 55 (line 137):
   $1 = nterm number ()
-> $$ = nterm value ()
Stack now 0 1 3 5 7 9 15 27 43 69 90 103

Entering state 109
Reading a token: Next token is token ';' ()
Reducing stack by rule 28 (line 87):
   $1 = token '=' ()
   $2 = nterm value ()
-> $$ = nterm init ()
Stack now 0 1 3 5 7 9 15 27 43 69 90
```

```
Entering state 104
Reducing stack by rule 22 (line 77):
   $1 = nterm declaration ()
   $2 = token IDENTIFIER ()
   $3 = token ':' ()
   $4 = nterm datatype ()
   $5 = nterm init ()
-> $$ = nterm statement ()
Stack now 0 1 3 5 7 9 15

Entering state 26
Next token is token ';' ()
Shifting token ';' ()

Entering state 42
Reducing stack by rule 12 (line 61):
   $1 = nterm statement ()
   $2 = token ';' ()
-> $$ = nterm block ()
Stack now 0 1 3 5 7 9 15

Entering state 25
Reading a token: Next token is token CONST ()
Shifting token CONST ()

Entering state 22
Reducing stack by rule 27 (line 84):
   $1 = token CONST ()
-> $$ = nterm declaration ()
Stack now 0 1 3 5 7 9 15 25

Entering state 27
Reading a token: Next token is token IDENTIFIER ()
Shifting token IDENTIFIER ()

Entering state 43
Reading a token: Next token is token ':' ()
Shifting token ':' ()

Entering state 69
Reading a token: Next token is token BOOLEANTYPE ()
Shifting token BOOLEANTYPE ()

Entering state 13
Reducing stack by rule 7 (line 52):
   $1 = token BOOLEANTYPE ()
-> $$ = nterm datatype ()
Stack now 0 1 3 5 7 9 15 25 27 43 69

Entering state 90
Reading a token: Next token is token '=' ()
Shifting token '=' ()
```

```
Entering state 103
Reading a token: Next token is token TRUE ()
Shifting token TRUE ()

Entering state 58
Reducing stack by rule 58 (line 140):
   $1 = token TRUE ()
-> $$ = nterm value ()
Stack now 0 1 3 5 7 9 15 25 27 43 69 90 103

Entering state 109
Reading a token: Next token is token ';' ()
Reducing stack by rule 28 (line 87):
   $1 = token '=' ()
   $2 = nterm value ()
-> $$ = nterm init ()
Stack now 0 1 3 5 7 9 15 25 27 43 69 90

Entering state 104
Reducing stack by rule 22 (line 77):
   $1 = nterm declaration ()
   $2 = token IDENTIFIER ()
   $3 = token ':' ()
   $4 = nterm datatype ()
   $5 = nterm init ()
-> $$ = nterm statement ()
Stack now 0 1 3 5 7 9 15 25

Entering state 26
Next token is token ';' ()
Shifting token ';' ()

Entering state 42
Reducing stack by rule 12 (line 61):
   $1 = nterm statement ()
   $2 = token ';' ()
-> $$ = nterm block ()
Stack now 0 1 3 5 7 9 15 25

Entering state 25
Reading a token: Next token is token VAR ()
Shifting token VAR ()

Entering state 21
Reducing stack by rule 26 (line 83):
   $1 = token VAR ()
-> $$ = nterm declaration ()
Stack now 0 1 3 5 7 9 15 25 25

Entering state 27
Reading a token: Next token is token IDENTIFIER ()
Shifting token IDENTIFIER ()
```

```
Entering state 43
Reading a token: Next token is token ':' ()
Shifting token ':' ()

Entering state 69
Reading a token: Next token is token NUMBERTYPE ()
Shifting token NUMBERTYPE ()

Entering state 11
Reducing stack by rule 5 (line 50):
   $1 = token NUMBERTYPE ()
-> $$ = nterm datatype ()
Stack now 0 1 3 5 7 9 15 25 25 27 43 69

Entering state 90
Reading a token: Next token is token '=' ()
Shifting token '=' ()

Entering state 103
Reading a token: Next token is token INTEGER ()
Shifting token INTEGER ()

Entering state 49
Reducing stack by rule 53 (line 133):
   $1 = token INTEGER ()
-> $$ = nterm number ()
Stack now 0 1 3 5 7 9 15 25 25 27 43 69 90 103

Entering state 54
Reducing stack by rule 55 (line 137):
   $1 = nterm number ()
-> $$ = nterm value ()
Stack now 0 1 3 5 7 9 15 25 25 27 43 69 90 103

Entering state 109
Reading a token: Next token is token MULT ()
Reducing stack by rule 38 (line 106):
   $1 = nterm value ()
-> $$ = nterm base ()
Stack now 0 1 3 5 7 9 15 25 25 27 43 69 90 103

Entering state 64
Next token is token MULT ()
Reducing stack by rule 37 (line 103):
   $1 = nterm base ()
-> $$ = nterm factor ()
Stack now 0 1 3 5 7 9 15 25 25 27 43 69 90 103

Entering state 63
Reducing stack by rule 34 (line 98):
   $1 = nterm factor ()
-> $$ = nterm term ()
Stack now 0 1 3 5 7 9 15 25 25 27 43 69 90 103
```

```
Entering state 62
Next token is token MULT ()
Shifting token MULT ()

Entering state 84
Reducing stack by rule 43 (line 117):
   $1 = token MULT ()
-> $$ = nterm mulops ()
Stack now 0 1 3 5 7 9 15 25 25 27 43 69 90 103 62

Entering state 86
Reading a token: Next token is token INTEGER ()
Shifting token INTEGER ()

Entering state 49
Reducing stack by rule 53 (line 133):
   $1 = token INTEGER ()
-> $$ = nterm number ()
Stack now 0 1 3 5 7 9 15 25 25 27 43 69 90 103 62 86

Entering state 54
Reducing stack by rule 55 (line 137):
   $1 = nterm number ()
-> $$ = nterm value ()
Stack now 0 1 3 5 7 9 15 25 25 27 43 69 90 103 62 86

Entering state 65
Reducing stack by rule 38 (line 106):
   $1 = nterm value ()
-> $$ = nterm base ()
Stack now 0 1 3 5 7 9 15 25 25 27 43 69 90 103 62 86

Entering state 64
Reading a token: Next token is token ';' ()
Reducing stack by rule 37 (line 103):
   $1 = nterm base ()
-> $$ = nterm factor ()
Stack now 0 1 3 5 7 9 15 25 25 27 43 69 90 103 62 86

Entering state 101
Reducing stack by rule 33 (line 97):
   $1 = nterm term ()
   $2 = nterm mulops ()
   $3 = nterm factor ()
-> $$ = nterm term ()
Stack now 0 1 3 5 7 9 15 25 25 27 43 69 90 103

Entering state 62
Next token is token ';' ()
Reducing stack by rule 32 (line 94):
   $1 = nterm term ()
-> $$ = nterm expression ()
Stack now 0 1 3 5 7 9 15 25 25 27 43 69 90 103
```

```
Entering state 108
Next token is token ';' ()
Reducing stack by rule 29 (line 88):
   $1 = token '=' ()
   $2 = nterm expression ()
-> $$ = nterm init ()
Stack now 0 1 3 5 7 9 15 25 25 27 43 69 90

Entering state 104
Reducing stack by rule 22 (line 77):
   $1 = nterm declaration ()
   $2 = token IDENTIFIER ()
   $3 = token ':' ()
   $4 = nterm datatype ()
   $5 = nterm init ()
-> $$ = nterm statement ()
Stack now 0 1 3 5 7 9 15 25 25

Entering state 26
Next token is token ';' ()
Shifting token ';' ()

Entering state 42
Reducing stack by rule 12 (line 61):
   $1 = nterm statement ()
   $2 = token ';' ()
-> $$ = nterm block ()
Stack now 0 1 3 5 7 9 15 25 25

Entering state 25
Reading a token: Next token is token IF ()
Shifting token IF ()

Entering state 18
Reading a token: Next token is token '(' ()
Shifting token '(' ()

Entering state 29
Reading a token: Next token is token IDENTIFIER ()
Shifting token IDENTIFIER ()

Entering state 51
Reducing stack by rule 56 (line 138):
   $1 = token IDENTIFIER ()
-> $$ = nterm value ()
Stack now 0 1 3 5 7 9 15 25 25 25 18 29

Entering state 55
Reading a token: Next token is token GT ()
Shifting token GT ()
```

```
Entering state 33
Reducing stack by rule 48 (line 126):
   $1 = token GT ()
-> $$ = nterm relop ()
Stack now 0 1 3 5 7 9 15 25 25 25 18 29 55

Entering state 77
Reading a token: Next token is token IDENTIFIER ()
Shifting token IDENTIFIER ()

Entering state 51
Reducing stack by rule 56 (line 138):
   $1 = token IDENTIFIER ()
-> $$ = nterm value ()
Stack now 0 1 3 5 7 9 15 25 25 25 18 29 55 77

Entering state 96
Reducing stack by rule 18 (line 71):
   $1 = nterm value ()
   $2 = nterm relop ()
   $3 = nterm value ()
-> $$ = nterm condition ()
Stack now 0 1 3 5 7 9 15 25 25 25 18 29

Entering state 53
Reading a token: Next token is token AND ()
Shifting token AND ()

Entering state 73
Reducing stack by rule 40 (line 110):
   $1 = token AND ()
-> $$ = nterm and_or ()
Stack now 0 1 3 5 7 9 15 25 25 25 18 29 53

Entering state 76
Reading a token: Next token is token IDENTIFIER ()
Shifting token IDENTIFIER ()

Entering state 51
Reducing stack by rule 56 (line 138):
   $1 = token IDENTIFIER ()
-> $$ = nterm value ()
Stack now 0 1 3 5 7 9 15 25 25 25 18 29 53 76

Entering state 55
Reading a token: Next token is token EQ ()
Shifting token EQ ()

Entering state 35
Reducing stack by rule 51 (line 129):
   $1 = token EQ ()
-> $$ = nterm relop ()
Stack now 0 1 3 5 7 9 15 25 25 25 18 29 53 76 55
```

```
Entering state 77
Reading a token: Next token is token TRUE ()
Shifting token TRUE ()

Entering state 58
Reducing stack by rule 58 (line 140):
   $1 = token TRUE ()
-> $$ = nterm value ()
Stack now 0 1 3 5 7 9 15 25 25 25 18 29 53 76 55 77

Entering state 96
Reducing stack by rule 18 (line 71):
   $1 = nterm value ()
   $2 = nterm relop ()
   $3 = nterm value ()
-> $$ = nterm condition ()
Stack now 0 1 3 5 7 9 15 25 25 25 18 29 53 76

Entering state 95
Reading a token: Next token is token ')' ()
Reducing stack by rule 17 (line 70):
   $1 = nterm condition ()
   $2 = nterm and_or ()
   $3 = nterm condition ()
-> $$ = nterm condition ()
Stack now 0 1 3 5 7 9 15 25 25 25 18 29

Entering state 53
Next token is token ')' ()
Shifting token ')' ()

Entering state 75
Reading a token: Next token is token '{' ()
Shifting token '{' ()

Entering state 94
Reading a token: Next token is token CONSOLELOG ()
Shifting token CONSOLELOG ()

Entering state 17
Reading a token: Next token is token '(' ()
Shifting token '(' ()

Entering state 28
Reading a token: Next token is token STRINGVALUE ()
Shifting token STRINGVALUE ()

Entering state 45
Reading a token: Next token is token ')' ()
Shifting token ')' ()
```

```
Entering state 71
Reading a token: Next token is token ';' ()
Shifting token ';' ()

Entering state 92
Reducing stack by rule 13 (line 62):
   $1 = token CONSOLELOG ()
   $2 = token '(' ()
   $3 = token STRINGVALUE ()
   $4 = token ')' ()
   $5 = token ';' ()
-> $$ = nterm block ()
Stack now 0 1 3 5 7 9 15 25 25 25 18 29 53 75 94

Entering state 25
Reading a token: Next token is token '}' ()
Reducing stack by rule 9 (line 56):
-> $$ = nterm body ()
Stack now 0 1 3 5 7 9 15 25 25 25 18 29 53 75 94 25

Entering state 41
Reducing stack by rule 8 (line 55):
   $1 = nterm block ()
   $2 = nterm body ()
-> $$ = nterm body ()
Stack now 0 1 3 5 7 9 15 25 25 25 18 29 53 75 94

Entering state 105
Next token is token '}' ()
Shifting token '}' ()

Entering state 110
Reading a token: Next token is token ELSE ()
Shifting token ELSE ()

Entering state 113
Reading a token: Next token is token '{' ()
Shifting token '{' ()

Entering state 116
Reading a token: Next token is token CONSOLELOG ()
Shifting token CONSOLELOG ()

Entering state 17
Reading a token: Next token is token '(' ()
Shifting token '(' ()

Entering state 28
Reading a token: Next token is token IDENTIFIER ()
Shifting token IDENTIFIER ()
```

```
Entering state 44
Reading a token: Next token is token ')' ()
Shifting token ')' ()

Entering state 70
Reading a token: Next token is token ';' ()
Shifting token ';' ()

Entering state 91
Reducing stack by rule 14 (line 63):
   $1 = token CONSOLELOG ()
   $2 = token '(' ()
   $3 = token IDENTIFIER ()
   $4 = token ')' ()
   $5 = token ';' ()
-> $$ = nterm block ()
Stack now 0 1 3 5 7 9 15 25 25 25 18 29 53 75 94 105 110 113 116

Entering state 25
Reading a token: Next token is token '}' ()
Reducing stack by rule 9 (line 56):
-> $$ = nterm body ()
Stack now 0 1 3 5 7 9 15 25 25 25 18 29 53 75 94 105 110 113 116 25

Entering state 41
Reducing stack by rule 8 (line 55):
   $1 = nterm block ()
   $2 = nterm body ()
-> $$ = nterm body ()
Stack now 0 1 3 5 7 9 15 25 25 25 18 29 53 75 94 105 110 113 116

Entering state 117
Next token is token '}' ()
Shifting token '}' ()

Entering state 118
Reducing stack by rule 15 (line 66):
   $1 = token ELSE ()
   $2 = token '{' ()
   $3 = nterm body ()
   $4 = token '}' ()
-> $$ = nterm else ()
Stack now 0 1 3 5 7 9 15 25 25 25 18 29 53 75 94 105 110

Entering state 114
Reducing stack by rule 11 (line 60):
   $1 = token IF ()
   $2 = token '(' ()
   $3 = nterm condition ()
   $4 = token ')' ()
   $5 = token '{' ()
   $6 = nterm body ()
   $7 = token '}' ()
```

```
   $8 = nterm else ()
-> $$ = nterm block ()
Stack now 0 1 3 5 7 9 15 25 25 25

Entering state 25
Reading a token: Next token is token LET ()
Shifting token LET ()

Entering state 20
Reducing stack by rule 25 (line 82):
   $1 = token LET ()
-> $$ = nterm declaration ()
Stack now 0 1 3 5 7 9 15 25 25 25 25

Entering state 27
Reading a token: Next token is token IDENTIFIER ()
Shifting token IDENTIFIER ()

Entering state 43
Reading a token: Next token is token ':' ()
Shifting token ':' ()

Entering state 69
Reading a token: Next token is token NUMBERTYPE ()
Shifting token NUMBERTYPE ()

Entering state 11
Reducing stack by rule 5 (line 50):
   $1 = token NUMBERTYPE ()
-> $$ = nterm datatype ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 27 43 69

Entering state 90
Reading a token: Next token is token '=' ()
Shifting token '=' ()

Entering state 103
Reading a token: Next token is token INTEGER ()
Shifting token INTEGER ()

Entering state 49
Reducing stack by rule 53 (line 133):
   $1 = token INTEGER ()
-> $$ = nterm number ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 27 43 69 90 103

Entering state 54
Reducing stack by rule 55 (line 137):
   $1 = nterm number ()
-> $$ = nterm value ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 27 43 69 90 103
```

```
Entering state 109
Reading a token: Next token is token ';' ()
Reducing stack by rule 28 (line 87):
   $1 = token '=' ()
   $2 = nterm value ()
-> $$ = nterm init ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 27 43 69 90

Entering state 104
Reducing stack by rule 22 (line 77):
   $1 = nterm declaration ()
   $2 = token IDENTIFIER ()
   $3 = token ':' ()
   $4 = nterm datatype ()
   $5 = nterm init ()
-> $$ = nterm statement ()
Stack now 0 1 3 5 7 9 15 25 25 25 25

Entering state 26
Next token is token ';' ()
Shifting token ';' ()

Entering state 42
Reducing stack by rule 12 (line 61):
   $1 = nterm statement ()
   $2 = token ';' ()
-> $$ = nterm block ()
Stack now 0 1 3 5 7 9 15 25 25 25 25

Entering state 25
Reading a token: Next token is token WHILE ()
Shifting token WHILE ()

Entering state 19
Reading a token: Next token is token '(' ()
Shifting token '(' ()

Entering state 30
Reading a token: Next token is token IDENTIFIER ()
Shifting token IDENTIFIER ()
Entering state 51
Reducing stack by rule 56 (line 138):
   $1 = token IDENTIFIER ()
-> $$ = nterm value ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 25 19 30

Entering state 55
Reading a token: Next token is token ')' ()
Reducing stack by rule 19 (line 72):
   $1 = nterm value ()
-> $$ = nterm condition ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 25 19 30
```

```
Entering state 56
Next token is token ')' ()
Shifting token ')' ()

Entering state 78
Reading a token: Next token is token '{' ()
Shifting token '{' ()

Entering state 97
Reading a token: Next token is token IDENTIFIER ()
Shifting token IDENTIFIER ()

Entering state 23
Reading a token: Next token is token '=' ()
Shifting token '=' ()

Entering state 37
Reading a token: Next token is token IDENTIFIER ()
Shifting token IDENTIFIER ()

Entering state 51
Reducing stack by rule 56 (line 138):
   $1 = token IDENTIFIER ()
-> $$ = nterm value ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 25 19 30 56 78 97 23 37

Entering state 65
Reducing stack by rule 38 (line 106):
   $1 = nterm value ()
-> $$ = nterm base ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 25 19 30 56 78 97 23 37

Entering state 64
Reading a token: Next token is token SUB ()
Reducing stack by rule 37 (line 103):
   $1 = nterm base ()
-> $$ = nterm factor ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 25 19 30 56 78 97 23 37

Entering state 63
Reducing stack by rule 34 (line 98):
   $1 = nterm factor ()
-> $$ = nterm term ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 25 19 30 56 78 97 23 37
Entering state 62
Next token is token SUB ()
Reducing stack by rule 32 (line 94):
   $1 = nterm term ()
-> $$ = nterm expression ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 25 19 30 56 78 97 23 37

Entering state 61
Next token is token SUB ()
```

```
Shifting token SUB ()

Entering state 82
Reducing stack by rule 46 (line 122):
   $1 = token SUB ()
-> $$ = nterm addops ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 25 19 30 56 78 97 23 37 61

Entering state 83
Reading a token: Next token is token INTEGER ()
Shifting token INTEGER ()

Entering state 49
Reducing stack by rule 53 (line 133):
   $1 = token INTEGER ()
-> $$ = nterm number ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 25 19 30 56 78 97 23 37 61 83

Entering state 54
Reducing stack by rule 55 (line 137):
   $1 = nterm number ()
-> $$ = nterm value ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 25 19 30 56 78 97 23 37 61 83

Entering state 65
Reducing stack by rule 38 (line 106):
   $1 = nterm value ()
-> $$ = nterm base ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 25 19 30 56 78 97 23 37 61 83

Entering state 64
Reading a token: Next token is token ';' ()
Reducing stack by rule 37 (line 103):
   $1 = nterm base ()
-> $$ = nterm factor ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 25 19 30 56 78 97 23 37 61 83

Entering state 63
Reducing stack by rule 34 (line 98):
   $1 = nterm factor ()
-> $$ = nterm term ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 25 19 30 56 78 97 23 37 61 83

Entering state 100
Next token is token ';' ()
Reducing stack by rule 31 (line 93):
   $1 = nterm expression ()
   $2 = nterm addops ()
   $3 = nterm term ()
-> $$ = nterm expression ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 25 19 30 56 78 97 23 37
```

```
Entering state 61
Next token is token ';' ()
Reducing stack by rule 23 (line 78):
   $1 = token IDENTIFIER ()
   $2 = token '=' ()
   $3 = nterm expression ()
-> $$ = nterm statement ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 25 19 30 56 78 97

Entering state 26
Next token is token ';' ()
Shifting token ';' ()

Entering state 42
Reducing stack by rule 12 (line 61):
   $1 = nterm statement ()
   $2 = token ';' ()
-> $$ = nterm block ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 25 19 30 56 78 97

Entering state 25
Reading a token: Next token is token '}' ()
Reducing stack by rule 9 (line 56):
-> $$ = nterm body ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 25 19 30 56 78 97 25

Entering state 41
Reducing stack by rule 8 (line 55):
   $1 = nterm block ()
   $2 = nterm body ()
-> $$ = nterm body ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 25 19 30 56 78 97

Entering state 106
Next token is token '}' ()
Shifting token '}' ()
Entering state 111
Reducing stack by rule 10 (line 59):
   $1 = token WHILE ()
   $2 = token '(' ()
   $3 = nterm condition ()
   $4 = token ')' ()
   $5 = token '{' ()
   $6 = nterm body ()
   $7 = token '}' ()
-> $$ = nterm block ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 25

Entering state 25
Reading a token: Next token is token RETURN ()
Reducing stack by rule 9 (line 56):
-> $$ = nterm body ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 25 25
```

```
Entering state 41
Reducing stack by rule 8 (line 55):
   $1 = nterm block ()
   $2 = nterm body ()
-> $$ = nterm body ()
Stack now 0 1 3 5 7 9 15 25 25 25 25 25

Entering state 41
Reducing stack by rule 8 (line 55):
   $1 = nterm block ()
   $2 = nterm body ()
-> $$ = nterm body ()
Stack now 0 1 3 5 7 9 15 25 25 25 25

Entering state 41
Reducing stack by rule 8 (line 55):
   $1 = nterm block ()
   $2 = nterm body ()
-> $$ = nterm body ()
Stack now 0 1 3 5 7 9 15 25 25 25

Entering state 41
Reducing stack by rule 8 (line 55):
   $1 = nterm block ()
   $2 = nterm body ()
-> $$ = nterm body ()
Stack now 0 1 3 5 7 9 15 25 25

Entering state 41
Reducing stack by rule 8 (line 55):
   $1 = nterm block ()
   $2 = nterm body ()
-> $$ = nterm body ()
Stack now 0 1 3 5 7 9 15 25

Entering state 41
Reducing stack by rule 8 (line 55):
   $1 = nterm block ()
   $2 = nterm body ()
-> $$ = nterm body ()
Stack now 0 1 3 5 7 9 15

Entering state 24
Next token is token RETURN ()
Shifting token RETURN ()

Entering state 39
Reading a token: Next token is token INTEGER ()
Shifting token INTEGER ()

Entering state 49
Reducing stack by rule 53 (line 133):
   $1 = token INTEGER ()
```

```
-> $$ = nterm number ()
Stack now 0 1 3 5 7 9 15 24 39

Entering state 54
Reducing stack by rule 55 (line 137):
   $1 = nterm number ()
-> $$ = nterm value ()
Stack now 0 1 3 5 7 9 15 24 39

Entering state 67
Reading a token: Next token is token ';' ()
Shifting token ';' ()

Entering state 89
Reducing stack by rule 61 (line 145):
   $1 = token RETURN ()
   $2 = nterm value ()
   $3 = token ';' ()
-> $$ = nterm return ()
Stack now 0 1 3 5 7 9 15 24

Entering state 40
Reading a token: Next token is token '}' ()
Shifting token '}' ()

Entering state 68
Reducing stack by rule 1 (line 42):
   $1 = token FUNCTION ()
   $2 = token IDENTIFIER ()
   $3 = token '(' ()
   $4 = nterm parameter ()
   $5 = token ')' ()
   $6 = token '{' ()
   $7 = nterm body ()
   $8 = nterm return ()
   $9 = token '}' ()
-> $$ = nterm function ()
Stack now 0

Entering state 2
Reading a token: Now at end of input.
Shifting token $end ()

Entering state 4
Stack now 0 2 4
Cleanup: popping token $end ()
Cleanup: popping nterm function ()
```

Terminal Output: Generated Parser Output

```
Terminals unused in grammar
    ELIF

Grammar
    0 $accept: function $end

    1 function: FUNCTION IDENTIFIER '(' parameter ')' '{' body return '}'

    2 parameter: parameter ',' parameter
    3          | IDENTIFIER ':' datatype
    4          | ε

    5 datatype: NUMBERTYPE
    6         | STRINGTYPE
    7         | BOOLEANTYPE

    8 body: block body
    9     | ε

   10 block: WHILE '(' condition ')' '{' body '}'
   11      | IF '(' condition ')' '{' body '}' else
   12      | statement ';'
   13      | CONSOLELOG '(' STRINGVALUE ')' ';'
   14      | CONSOLELOG '(' IDENTIFIER ')' ';'

   15 else: ELSE '{' body '}'
   16     | ε

   17 condition: condition and_or condition
   18          | value relop value
   19          | value
   20          | TRUE
   21          | FALSE

   22 statement: declaration IDENTIFIER ':' datatype init
   23          | IDENTIFIER '=' expression
   24          | IDENTIFIER relop expression

   25 declaration: LET
   26            | VAR
   27            | CONST

   28 init: '=' value
   29     | '=' expression
   30     | ε

   31 expression: expression addops term
   32           | term

   33 term: term mulops factor
   34     | factor
```

```
   35 factor: base exponent base
   36       | LOG '(' value ',' value ')'
   37       | base

   38 base: value
   39     | '(' expression ')'

   40 and_or: AND
   41       | OR

   42 exponent: POW

   43 mulops: MULT
   44       | DIV

   45 addops: ADD
   46       | SUB

   47 relop: LT
   48      | GT
   49      | LE
   50      | GE
   51      | EQ
   52      | NE

   53 number: INTEGER
   54       | FLOAT

   55 value: number
   56      | IDENTIFIER
   57      | STRINGVALUE
   58      | TRUE
   59      | FALSE
   60      | SCAN '(' ')'

   61 return: RETURN value ';'
   62       | ε


Terminals, with rules where they appear

    $end (0) 0
    '(' (40) 1 10 11 13 14 36 39 60
    ')' (41) 1 10 11 13 14 36 39 60
    ',' (44) 2 36
    ':' (58) 3 22
    ';' (59) 12 13 14 61
    '=' (61) 23 28 29
    '{' (123) 1 10 11 15
    '}' (125) 1 10 11 15
    error (256)
    CONSOLELOG <treeNode> (258) 13 14
    SCAN <treeNode> (259) 60
```

```
    IF <treeNode> (260) 11
    WHILE <treeNode> (261) 10
    ELSE <treeNode> (262) 15
    RETURN <treeNode> (263) 61
    ELIF <treeNode> (264)
    LET <treeNode> (265) 25
    VAR <treeNode> (266) 26
    CONST <treeNode> (267) 27
    ADD <treeNode> (268) 45
    SUB <treeNode> (269) 46
    MULT <treeNode> (270) 43
    DIV <treeNode> (271) 44
    LOG <treeNode> (272) 36
    POW <treeNode> (273) 42
    GE <treeNode> (274) 50
    LE <treeNode> (275) 49
    GT <treeNode> (276) 48
    LT <treeNode> (277) 47
    EQ <treeNode> (278) 51
    NE <treeNode> (279) 52
    TRUE <treeNode> (280) 20 58
    FALSE <treeNode> (281) 21 59
    AND <treeNode> (282) 40
    OR <treeNode> (283) 41
    NUMBERTYPE <treeNode> (284) 5
    STRINGTYPE <treeNode> (285) 6
    BOOLEANTYPE <treeNode> (286) 7
    FUNCTION <treeNode> (287) 1
    INTEGER <treeNode> (288) 53
    FLOAT <treeNode> (289) 54
    IDENTIFIER <treeNode> (290) 1 3 14 22 23 24 56
    STRINGVALUE <treeNode> (291) 13 57

Nonterminals, with rules where they appear

    $accept (45)
        on left: 0
    function <treeNode> (46)
        on left: 1
        on right: 0
    parameter <treeNode> (47)
        on left: 2 3 4
        on right: 1 2
    datatype <treeNode> (48)
        on left: 5 6 7
        on right: 3 22
    body <treeNode> (49)
        on left: 8 9
        on right: 1 8 10 11 15
    block <treeNode> (50)
        on left: 10 11 12 13 14
        on right: 8
    else <treeNode> (51)
```

```
    on left: 15 16
    on right: 11
condition <treeNode> (52)
    on left: 17 18 19 20 21
    on right: 10 11 17
statement <treeNode> (53)
    on left: 22 23 24
    on right: 12
declaration <treeNode> (54)
    on left: 25 26 27
    on right: 22
init <treeNode> (55)
    on left: 28 29 30
    on right: 22
expression <treeNode> (56)
    on left: 31 32
    on right: 23 24 29 31 39
term <treeNode> (57)
    on left: 33 34
    on right: 31 32 33
factor <treeNode> (58)
    on left: 35 36 37
    on right: 33 34
base <treeNode> (59)
    on left: 38 39
    on right: 35 37
and_or <treeNode> (60)
    on left: 40 41
    on right: 17
exponent <treeNode> (61)
    on left: 42
    on right: 35
mulops <treeNode> (62)
    on left: 43 44
    on right: 33
addops <treeNode> (63)
    on left: 45 46
    on right: 31
relop <treeNode> (64)
    on left: 47 48 49 50 51 52
    on right: 18 24
number <treeNode> (65)
    on left: 53 54
    on right: 55
value <treeNode> (66)
    on left: 55 56 57 58 59 60
    on right: 18 19 28 36 38 61
return <treeNode> (67)
    on left: 61 62
    on right: 1
```

```
State 0
    0 $accept: • function $end
    FUNCTION  shift, and go to state 1
    function  go to state 2

State 1
    1 function: FUNCTION • IDENTIFIER '(' parameter ')' '{' body return '}'
    IDENTIFIER  shift, and go to state 3

State 2
    0 $accept: function • $end
    $end  shift, and go to state 4

State 3
    1 function: FUNCTION IDENTIFIER • '(' parameter ')' '{' body return '}'
    '('  shift, and go to state 5

State 4
    0 $accept: function $end •
    $default  accept

State 5
    1 function: FUNCTION IDENTIFIER '(' • parameter ')' '{' body return '}'
    IDENTIFIER  shift, and go to state 6
    $default  reduce using rule 4 (parameter)
    parameter  go to state 7

State 6
    3 parameter: IDENTIFIER • ':' datatype
    ':'  shift, and go to state 8

State 7
    1 function: FUNCTION IDENTIFIER '(' parameter • ')' '{' body return '}'
    2 parameter: parameter • ',' parameter
    ')'  shift, and go to state 9
    ','  shift, and go to state 10

State 8
    3 parameter: IDENTIFIER ':' • datatype
    NUMBERTYPE   shift, and go to state 11
    STRINGTYPE   shift, and go to state 12
    BOOLEANTYPE  shift, and go to state 13
    datatype  go to state 14

State 9
    1 function: FUNCTION IDENTIFIER '(' parameter ')' • '{' body return '}'
    '{'  shift, and go to state 15

State 10
    2 parameter: parameter ',' • parameter
    IDENTIFIER  shift, and go to state 6
    $default  reduce using rule 4 (parameter)
    parameter  go to state 16
```

```
State 11
    5 datatype: NUMBERTYPE •
    $default  reduce using rule 5 (datatype)

State 12
    6 datatype: STRINGTYPE •
    $default  reduce using rule 6 (datatype)

State 13
    7 datatype: BOOLEANTYPE •
    $default  reduce using rule 7 (datatype)

State 14
    3 parameter: IDENTIFIER ':' datatype •
    $default  reduce using rule 3 (parameter)

State 15
    1 function: FUNCTION IDENTIFIER '(' parameter ')' '{' • body return '}'
    CONSOLELOG  shift, and go to state 17
    IF          shift, and go to state 18
    WHILE       shift, and go to state 19
    LET         shift, and go to state 20
    VAR         shift, and go to state 21
    CONST       shift, and go to state 22
    IDENTIFIER  shift, and go to state 23
    $default  reduce using rule 9 (body)
    body         go to state 24
    block        go to state 25
    statement    go to state 26
    declaration  go to state 27

State 16
    2 parameter: parameter • ',' parameter
    2          | parameter ',' parameter •
    ','  shift, and go to state 10
    ','       [reduce using rule 2 (parameter)]
    $default  reduce using rule 2 (parameter)

State 17
   13 block: CONSOLELOG • '(' STRINGVALUE ')' ';'
   14      | CONSOLELOG • '(' IDENTIFIER ')' ';'
    '('  shift, and go to state 28

State 18
   11 block: IF • '(' condition ')' '{' body '}' else
    '('  shift, and go to state 29

State 19
   10 block: WHILE • '(' condition ')' '{' body '}'
    '('  shift, and go to state 30

State 20
   25 declaration: LET •
```

```
    $default  reduce using rule 25 (declaration)

State 21
   26 declaration: VAR •
    $default  reduce using rule 26 (declaration)

State 22
   27 declaration: CONST •
    $default  reduce using rule 27 (declaration)

State 23
   23 statement: IDENTIFIER • '=' expression
   24          | IDENTIFIER • relop expression
    GE   shift, and go to state 31
    LE   shift, and go to state 32
    GT   shift, and go to state 33
    LT   shift, and go to state 34
    EQ   shift, and go to state 35
    NE   shift, and go to state 36
    '='  shift, and go to state 37
    relop  go to state 38

State 24
    1 function: FUNCTION IDENTIFIER '(' parameter ')' '{' body • return '}'
    RETURN  shift, and go to state 39
    $default  reduce using rule 62 (return)
    return  go to state 40

State 25
    8 body: block • body
    CONSOLELOG  shift, and go to state 17
    IF          shift, and go to state 18
    WHILE       shift, and go to state 19
    LET         shift, and go to state 20
    VAR         shift, and go to state 21
    CONST       shift, and go to state 22
    IDENTIFIER  shift, and go to state 23
    $default  reduce using rule 9 (body)
    body         go to state 41
    block        go to state 25
    statement    go to state 26
    declaration  go to state 27

State 26
   12 block: statement • ';'
    ';'  shift, and go to state 42

State 27
   22 statement: declaration • IDENTIFIER ':' datatype init
    IDENTIFIER  shift, and go to state 43

State 28
   13 block: CONSOLELOG '(' • STRINGVALUE ')' ';'
```

```
   14      | CONSOLELOG '(' • IDENTIFIER ')' ';'
    IDENTIFIER   shift, and go to state 44
    STRINGVALUE  shift, and go to state 45

State 29
   11 block: IF '(' • condition ')' '{' body '}' else
    SCAN         shift, and go to state 46
    TRUE         shift, and go to state 47
    FALSE        shift, and go to state 48
    INTEGER      shift, and go to state 49
    FLOAT        shift, and go to state 50
    IDENTIFIER   shift, and go to state 51
    STRINGVALUE  shift, and go to state 52
    condition  go to state 53
    number     go to state 54
    value      go to state 55

State 30
   10 block: WHILE '(' • condition ')' '{' body '}'
    SCAN         shift, and go to state 46
    TRUE         shift, and go to state 47
    FALSE        shift, and go to state 48
    INTEGER      shift, and go to state 49
    FLOAT        shift, and go to state 50
    IDENTIFIER   shift, and go to state 51
    STRINGVALUE  shift, and go to state 52
    condition  go to state 56
    number     go to state 54
    value      go to state 55

State 31
   50 relop: GE •
    $default  reduce using rule 50 (relop)

State 32
   49 relop: LE •
    $default  reduce using rule 49 (relop)

State 33
   48 relop: GT •
    $default  reduce using rule 48 (relop)

State 34
   47 relop: LT •
    $default  reduce using rule 47 (relop)

State 35
   51 relop: EQ •
    $default  reduce using rule 51 (relop)

State 36
   52 relop: NE •
    $default  reduce using rule 52 (relop)
```

```
State 37
   23 statement: IDENTIFIER '=' • expression
    SCAN         shift, and go to state 46
    LOG          shift, and go to state 57
    TRUE         shift, and go to state 58
    FALSE        shift, and go to state 59
    INTEGER      shift, and go to state 49
    FLOAT        shift, and go to state 50
    IDENTIFIER   shift, and go to state 51
    STRINGVALUE  shift, and go to state 52
    '('          shift, and go to state 60
    expression  go to state 61
    term        go to state 62
    factor      go to state 63
    base        go to state 64
    number      go to state 54
    value       go to state 65

State 38
   24 statement: IDENTIFIER relop • expression
    SCAN         shift, and go to state 46
    LOG          shift, and go to state 57
    TRUE         shift, and go to state 58
    FALSE        shift, and go to state 59
    INTEGER      shift, and go to state 49
    FLOAT        shift, and go to state 50
    IDENTIFIER   shift, and go to state 51
    STRINGVALUE  shift, and go to state 52
    '('          shift, and go to state 60
    expression  go to state 66
    term        go to state 62
    factor      go to state 63
    base        go to state 64
    number      go to state 54
    value       go to state 65

State 39
   61 return: RETURN • value ';'
    SCAN         shift, and go to state 46
    TRUE         shift, and go to state 58
    FALSE        shift, and go to state 59
    INTEGER      shift, and go to state 49
    FLOAT        shift, and go to state 50
    IDENTIFIER   shift, and go to state 51
    STRINGVALUE  shift, and go to state 52
    number  go to state 54
    value   go to state 67

State 40
    1 function: FUNCTION IDENTIFIER '(' parameter ')' '{' body return • '}'
    '}'  shift, and go to state 68
```

```
State 41
    8 body: block body •
    $default  reduce using rule 8 (body)

State 42
   12 block: statement ';' •
    $default  reduce using rule 12 (block)

State 43
   22 statement: declaration IDENTIFIER • ':' datatype init
    ':'  shift, and go to state 69

State 44
   14 block: CONSOLELOG '(' IDENTIFIER • ')' ';'
    ')'  shift, and go to state 70

State 45
   13 block: CONSOLELOG '(' STRINGVALUE • ')' ';'
    ')'  shift, and go to state 71

State 46
   60 value: SCAN • '(' ')'
    '('  shift, and go to state 72

State 47
   20 condition: TRUE •
   58 value: TRUE •
    AND       reduce using rule 20 (condition)
    AND       [reduce using rule 58 (value)]
    OR        reduce using rule 20 (condition)
    OR        [reduce using rule 58 (value)]
    ')'       reduce using rule 20 (condition)
    ')'       [reduce using rule 58 (value)]
    $default  reduce using rule 58 (value)

State 48
   21 condition: FALSE •
   59 value: FALSE •
    AND       reduce using rule 21 (condition)
    AND       [reduce using rule 59 (value)]
    OR        reduce using rule 21 (condition)
    OR        [reduce using rule 59 (value)]
    ')'       reduce using rule 21 (condition)
    ')'       [reduce using rule 59 (value)]
    $default  reduce using rule 59 (value)

State 49
   53 number: INTEGER •
    $default  reduce using rule 53 (number)

State 50
   54 number: FLOAT •
    $default  reduce using rule 54 (number)
```

```
State 51
   56 value: IDENTIFIER •
    $default  reduce using rule 56 (value)

State 52
   57 value: STRINGVALUE •
    $default  reduce using rule 57 (value)

State 53
   11 block: IF '(' condition • ')' '{' body '}' else
   17 condition: condition • and_or condition
    AND  shift, and go to state 73
    OR   shift, and go to state 74
    ')'  shift, and go to state 75
    and_or  go to state 76

State 54
   55 value: number •
    $default  reduce using rule 55 (value)

State 55
   18 condition: value • relop value
   19          | value •
    GE  shift, and go to state 31
    LE  shift, and go to state 32
    GT  shift, and go to state 33
    LT  shift, and go to state 34
    EQ  shift, and go to state 35
    NE  shift, and go to state 36
    $default  reduce using rule 19 (condition)
    relop  go to state 77

State 56
   10 block: WHILE '(' condition • ')' '{' body '}'
   17 condition: condition • and_or condition
    AND  shift, and go to state 73
    OR   shift, and go to state 74
    ')'  shift, and go to state 78
    and_or  go to state 76

State 57
   36 factor: LOG • '(' value ',' value ')'
    '('  shift, and go to state 79

State 58
   58 value: TRUE •
    $default  reduce using rule 58 (value)

State 59
   59 value: FALSE •
    $default  reduce using rule 59 (value)
```

```
State 60
   39 base: '(' • expression ')'
    SCAN         shift, and go to state 46
    LOG          shift, and go to state 57
    TRUE         shift, and go to state 58
    FALSE        shift, and go to state 59
    INTEGER      shift, and go to state 49
    FLOAT        shift, and go to state 50
    IDENTIFIER   shift, and go to state 51
    STRINGVALUE  shift, and go to state 52
    '('          shift, and go to state 60
    expression  go to state 80
    term        go to state 62
    factor      go to state 63
    base        go to state 64
    number      go to state 54
    value       go to state 65

State 61
   23 statement: IDENTIFIER '=' expression •
   31 expression: expression • addops term
    ADD  shift, and go to state 81
    SUB  shift, and go to state 82
    $default  reduce using rule 23 (statement)
    addops  go to state 83

State 62
   32 expression: term •
   33 term: term • mulops factor
    MULT  shift, and go to state 84
    DIV   shift, and go to state 85
    $default  reduce using rule 32 (expression)
    mulops  go to state 86

State 63
   34 term: factor •
    $default  reduce using rule 34 (term)

State 64
   35 factor: base • exponent base
   37       | base •
    POW  shift, and go to state 87
    $default  reduce using rule 37 (factor)
    exponent  go to state 88

State 65
   38 base: value •
    $default  reduce using rule 38 (base)

State 66
   24 statement: IDENTIFIER relop expression •
   31 expression: expression • addops term
    ADD  shift, and go to state 81
```

```
    SUB  shift, and go to state 82
    $default  reduce using rule 24 (statement)
    addops  go to state 83

State 67
   61 return: RETURN value • ';'
    ';'  shift, and go to state 89

State 68
    1 function: FUNCTION IDENTIFIER '(' parameter ')' '{' body return '}' •
    $default  reduce using rule 1 (function)

State 69
   22 statement: declaration IDENTIFIER ':' • datatype init
    NUMBERTYPE   shift, and go to state 11
    STRINGTYPE   shift, and go to state 12
    BOOLEANTYPE  shift, and go to state 13
    datatype  go to state 90

State 70
   14 block: CONSOLELOG '(' IDENTIFIER ')' • ';'
    ';'  shift, and go to state 91

State 71
   13 block: CONSOLELOG '(' STRINGVALUE ')' • ';'
    ';'  shift, and go to state 92

State 72
   60 value: SCAN '(' • ')'
    ')'  shift, and go to state 93

State 73
   40 and_or: AND •
    $default  reduce using rule 40 (and_or)

State 74
   41 and_or: OR •
    $default  reduce using rule 41 (and_or)

State 75
   11 block: IF '(' condition ')' • '{' body '}' else
    '{'  shift, and go to state 94

State 76
   17 condition: condition and_or • condition
    SCAN         shift, and go to state 46
    TRUE         shift, and go to state 47
    FALSE        shift, and go to state 48
    INTEGER      shift, and go to state 49
    FLOAT        shift, and go to state 50
    IDENTIFIER   shift, and go to state 51
    STRINGVALUE  shift, and go to state 52
    condition  go to state 95
```

```
    number     go to state 54
    value      go to state 55

State 77
   18 condition: value relop • value
    SCAN         shift, and go to state 46
    TRUE         shift, and go to state 58
    FALSE        shift, and go to state 59
    INTEGER      shift, and go to state 49
    FLOAT        shift, and go to state 50
    IDENTIFIER   shift, and go to state 51
    STRINGVALUE  shift, and go to state 52
    number  go to state 54
    value   go to state 96

State 78
   10 block: WHILE '(' condition ')' • '{' body '}'
    '{'  shift, and go to state 97

State 79
   36 factor: LOG '(' • value ',' value ')'
    SCAN         shift, and go to state 46
    TRUE         shift, and go to state 58
    FALSE        shift, and go to state 59
    INTEGER      shift, and go to state 49
    FLOAT        shift, and go to state 50
    IDENTIFIER   shift, and go to state 51
    STRINGVALUE  shift, and go to state 52
    number  go to state 54
    value   go to state 98

State 80
   31 expression: expression • addops term
   39 base: '(' expression • ')'
    ADD  shift, and go to state 81
    SUB  shift, and go to state 82
    ')'  shift, and go to state 99
    addops  go to state 83

State 81
   45 addops: ADD •
    $default  reduce using rule 45 (addops)

State 82
   46 addops: SUB •
    $default  reduce using rule 46 (addops)

State 83
   31 expression: expression addops • term
    SCAN         shift, and go to state 46
    LOG          shift, and go to state 57
    TRUE         shift, and go to state 58
    FALSE        shift, and go to state 59
```

```
    INTEGER      shift, and go to state 49
    FLOAT        shift, and go to state 50
    IDENTIFIER   shift, and go to state 51
    STRINGVALUE  shift, and go to state 52
    '('          shift, and go to state 60
    term    go to state 100
    factor  go to state 63
    base    go to state 64
    number  go to state 54
    value   go to state 65

State 84
   43 mulops: MULT •
    $default  reduce using rule 43 (mulops)

State 85
   44 mulops: DIV •
    $default  reduce using rule 44 (mulops)

State 86
   33 term: term mulops • factor
    SCAN         shift, and go to state 46
    LOG          shift, and go to state 57
    TRUE         shift, and go to state 58
    FALSE        shift, and go to state 59
    INTEGER      shift, and go to state 49
    FLOAT        shift, and go to state 50
    IDENTIFIER   shift, and go to state 51
    STRINGVALUE  shift, and go to state 52
    '('          shift, and go to state 60
    factor  go to state 101
    base    go to state 64
    number  go to state 54
    value   go to state 65

State 87
   42 exponent: POW •
    $default  reduce using rule 42 (exponent)

State 88
   35 factor: base exponent • base
    SCAN         shift, and go to state 46
    TRUE         shift, and go to state 58
    FALSE        shift, and go to state 59
    INTEGER      shift, and go to state 49
    FLOAT        shift, and go to state 50
    IDENTIFIER   shift, and go to state 51
    STRINGVALUE  shift, and go to state 52
    '('          shift, and go to state 60
    base    go to state 102
    number  go to state 54
    value   go to state 65
```

```
State 89
   61 return: RETURN value ';' •
    $default  reduce using rule 61 (return)

State 90
   22 statement: declaration IDENTIFIER ':' datatype • init
    '='  shift, and go to state 103
    $default  reduce using rule 30 (init)
    init  go to state 104

State 91
   14 block: CONSOLELOG '(' IDENTIFIER ')' ';' •
    $default  reduce using rule 14 (block)

State 92
   13 block: CONSOLELOG '(' STRINGVALUE ')' ';' •
    $default  reduce using rule 13 (block)

State 93
   60 value: SCAN '(' ')' •
    $default  reduce using rule 60 (value)

State 94
   11 block: IF '(' condition ')' '{' • body '}' else
    CONSOLELOG  shift, and go to state 17
    IF          shift, and go to state 18
    WHILE       shift, and go to state 19
    LET         shift, and go to state 20
    VAR         shift, and go to state 21
    CONST       shift, and go to state 22
    IDENTIFIER  shift, and go to state 23
    $default  reduce using rule 9 (body)
    body         go to state 105
    block        go to state 25
    statement    go to state 26
    declaration  go to state 27

State 95
   17 condition: condition • and_or condition
   17          | condition and_or condition •
    AND  shift, and go to state 73
    OR   shift, and go to state 74
    AND       [reduce using rule 17 (condition)]
    OR        [reduce using rule 17 (condition)]
    $default  reduce using rule 17 (condition)
    and_or  go to state 76

State 96
   18 condition: value relop value •
    $default  reduce using rule 18 (condition)

State 97
   10 block: WHILE '(' condition ')' '{' • body '}'
```

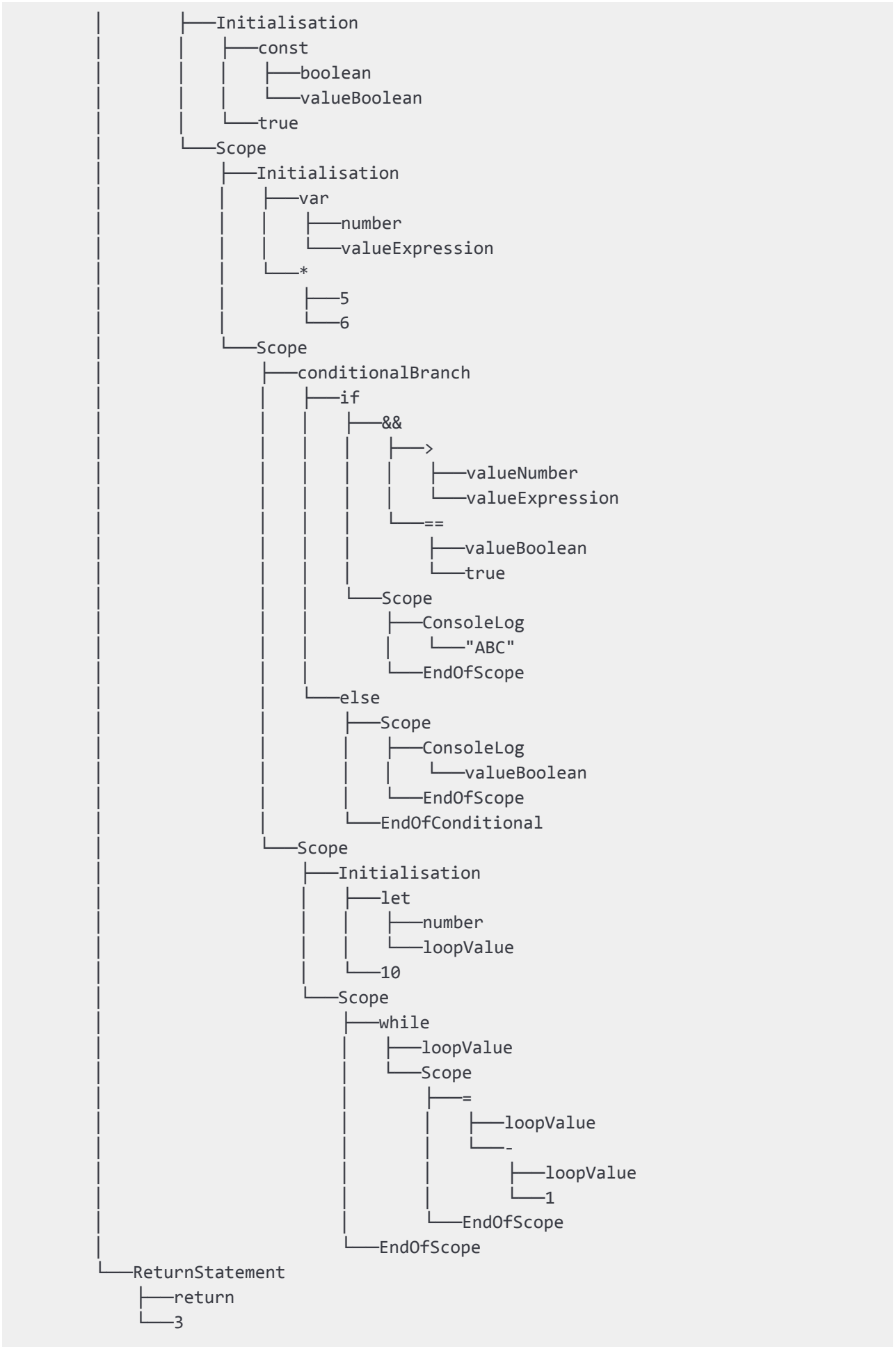
```
    CONSOLELOG  shift, and go to state 17
    IF          shift, and go to state 18
    WHILE       shift, and go to state 19
    LET         shift, and go to state 20
    VAR         shift, and go to state 21
    CONST       shift, and go to state 22
    IDENTIFIER  shift, and go to state 23
    $default  reduce using rule 9 (body)
    body         go to state 106
    block        go to state 25
    statement    go to state 26
    declaration  go to state 27

State 98
   36 factor: LOG '(' value • ',' value ')'
    ','  shift, and go to state 107

State 99
   39 base: '(' expression ')' •
    $default  reduce using rule 39 (base)

State 100
   31 expression: expression addops term •
   33 term: term • mulops factor
    MULT  shift, and go to state 84
    DIV   shift, and go to state 85
    $default  reduce using rule 31 (expression)
    mulops  go to state 86

State 101
   33 term: term mulops factor •
    $default  reduce using rule 33 (term)

State 102
   35 factor: base exponent base •
    $default  reduce using rule 35 (factor)

State 103
   28 init: '=' • value
   29     | '=' • expression
    SCAN         shift, and go to state 46
    LOG          shift, and go to state 57
    TRUE         shift, and go to state 58
    FALSE        shift, and go to state 59
    INTEGER      shift, and go to state 49
    FLOAT        shift, and go to state 50
    IDENTIFIER   shift, and go to state 51
    STRINGVALUE  shift, and go to state 52
    '('          shift, and go to state 60
    expression  go to state 108
    term        go to state 62
    factor      go to state 63
    base        go to state 64
```

```
    number      go to state 54
    value       go to state 109

State 104
   22 statement: declaration IDENTIFIER ':' datatype init •
    $default  reduce using rule 22 (statement)

State 105
   11 block: IF '(' condition ')' '{' body • '}' else
    '}'  shift, and go to state 110

State 106
   10 block: WHILE '(' condition ')' '{' body • '}'
    '}'  shift, and go to state 111

State 107
   36 factor: LOG '(' value ',' • value ')'
    SCAN         shift, and go to state 46
    TRUE         shift, and go to state 58
    FALSE        shift, and go to state 59
    INTEGER      shift, and go to state 49
    FLOAT        shift, and go to state 50
    IDENTIFIER   shift, and go to state 51
    STRINGVALUE  shift, and go to state 52
    number  go to state 54
    value   go to state 112

State 108
   29 init: '=' expression •
   31 expression: expression • addops term
    ADD  shift, and go to state 81
    SUB  shift, and go to state 82
    $default  reduce using rule 29 (init)
    addops  go to state 83

State 109
   28 init: '=' value •
   38 base: value •
    ';'       reduce using rule 28 (init)
    ';'       [reduce using rule 38 (base)]
    $default  reduce using rule 38 (base)

State 110
   11 block: IF '(' condition ')' '{' body '}' • else
    ELSE  shift, and go to state 113
    $default  reduce using rule 16 (else)
    else  go to state 114

State 111
   10 block: WHILE '(' condition ')' '{' body '}' •
    $default  reduce using rule 10 (block)
```

```
State 112
   36 factor: LOG '(' value ',' value • ')'
    ')'  shift, and go to state 115

State 113
   15 else: ELSE • '{' body '}'
    '{'  shift, and go to state 116

State 114
   11 block: IF '(' condition ')' '{' body '}' else •
    $default  reduce using rule 11 (block)

State 115
   36 factor: LOG '(' value ',' value ')' •
    $default  reduce using rule 36 (factor)

State 116
   15 else: ELSE '{' • body '}'
    CONSOLELOG  shift, and go to state 17
    IF          shift, and go to state 18
    WHILE       shift, and go to state 19
    LET         shift, and go to state 20
    VAR         shift, and go to state 21
    CONST       shift, and go to state 22
    IDENTIFIER  shift, and go to state 23
    $default  reduce using rule 9 (body)
    body         go to state 117
    block        go to state 25
    statement    go to state 26
    declaration  go to state 27

State 117
   15 else: ELSE '{' body • '}'
    '}'  shift, and go to state 118

State 118
   15 else: ELSE '{' body '}' •
    $default  reduce using rule 15 (else)
```

Terminal Output: Generated Parse Tree

```
Parse Tree

└──Function
    └──trialFunction
        ├──Scope
        │   ├──Initialisation
        │   │   ├──let
        │   │   │   ├──number
        │   │   │   └──valueNumber
        │   │   └──3
        │   └──Scope
```


Initialisation
const
boolean
valueBoolean
true
Scope
Initialisation
var
number
valueExpression
*
5
6
Scope
conditionalBranch
if
&&
>
valueNumber
valueExpression
==
valueBoolean
true
Scope
ConsoleLog
"ABC"
EndOfScope
else
Scope
ConsoleLog
valueBoolean
EndOfScope
EndOfConditional
Scope
Initialisation
let
number
loopValue
10
Scope
while
loopValue
Scope
=
loopValue
-
loopValue
1
EndOfScope
EndOfScope
ReturnStatement
return
3

IV. Error Handling with Terminal Output

Input Program (with Unrecognised Symbols)

```
elif (valueBoolean == true) {
    console.log(valueBoolean);
}
```

Terminal Output

```
Parsing Failed
Line Number: 1,syntax error, unexpected '('
```

Input Program (with Unmatched Braces)

```
function errorFunction() {
```

Terminal Output

```
Parsing Failed
Line Number: 1,syntax error, unexpected FUNCTION, expecting end of file
```

Input Program (with Invalid Identifiers)

```
let 123valueString: string = "Dru";
let valueNumber: number = 789;
let valueFloat: number = 0.123_456;
```

Terminal Output

```
Parsing Failed
Line Number: 1,syntax error, unexpected INTEGER, expecting IDENTIFIER
```

Input Program (with Invalid Expressions)

```
let valueNumber: number = 3;
const valueBoolean: boolean = true;
var valueExpression: number = 5 +* 6;
```

Terminal Output

```
Parsing Failed
Line Number: 3,syntax error, unexpected MULT
```