<u>Syntax Directed Translation Design</u>

Grammar and Semantic Rule

Non-Terminals	**Terminals**
S	program (keyword)
BEGIN_BLOCK	begin (keyword)
BEGIN_STMT	end (keyword)
STMT	int (keyword)
DECLARATION	identifier
OPREATOR	number_literal
OP_STMT	operators (=, *)
FACTOR	separator (;, ,)
PRINT_CONTENT	string_literal
PRINT_STMT	

Production	**Semantic rules**
S -> BEGIN_BLOCK BEGIN_STMT	Initialize Symbol Table
BEGIN_BLOCK -> program id	BEGIN_BLOCK.program = id.lexval
BEGIN_STMT -> begin STMT* end	BEGIN_STMT.program = begin.lexval + STMT.program + end.lexval
STMT -> DECLARATION	STMT.inh = DECLARATION.inh
STMT -> PRINT_STMT	STMT.inh = PRINT_STMT.val

DECLARATION -> int OPERATOR ;	DECLARATION.inh = int.lexval
	addType(DECLARATION.inh, OPERATOR.val)

DECLARATION -> OPERATOR ,	addType(DECLARATION.inh, OPERATOR.val)

OPERATOR -> id = OP_STMT	OPERATOR.inh = OP_STMT.inh
	addToSymbolTable(id.lexval, OPERATOR.inh)

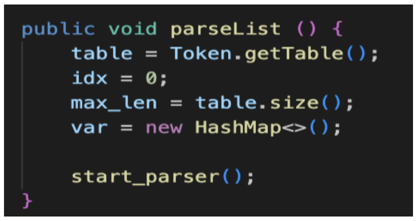
OPERATOR -> id	OPERATOR.val = id.lexval
	addToSymbolTable(id.lexval, null)

OP_STMT -> FACTOR	OP_STMT.val = FACTOR.val
OP_STMT -> OP_STMT * FACTOR	OP_STMT.val = OP_STMT.val * FACTOR.val (compute calculations from inherited values)
FACTOR -> number_literal	FACTOR.val = number_literal.lexval(converted into type)
FACTOR -> id	FACTOR.val = id.lexval
PRINT_CONTENT -> string_literal	PRINT_CONTENT.val = string_literal.lexval
PRINT_CONTENT -> id	PRINT_CONTENT.val = id.val

PRINT_STMT -> print_line (PRINT_CONTENT) ;	Print output (PRINT_CONTENT.val)

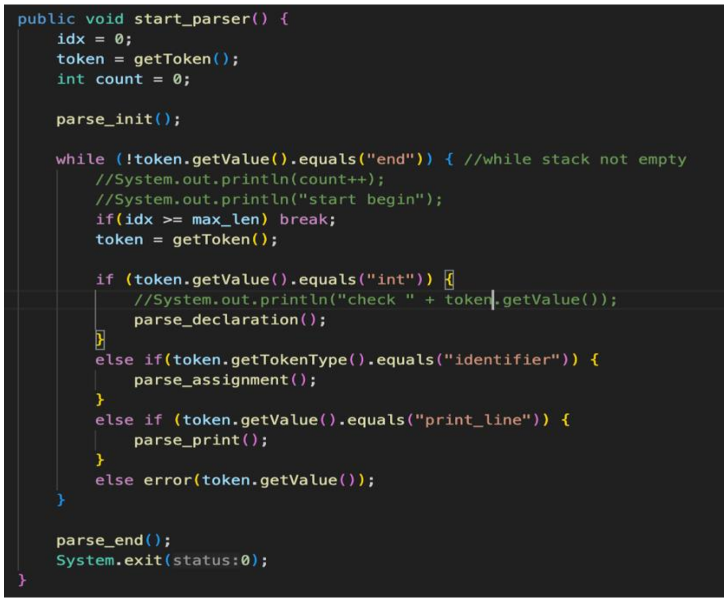
Explanation of the Source Code

Initialization of Symbol List “var”



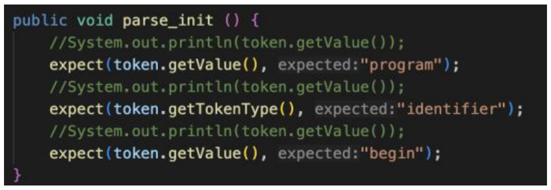
```java
public void parseList () {
    table = Token.getTable();
    idx = 0;
    max_len = table.size();
    var = new HashMap<>();

    start_parser();
}
```

At starting terminal S



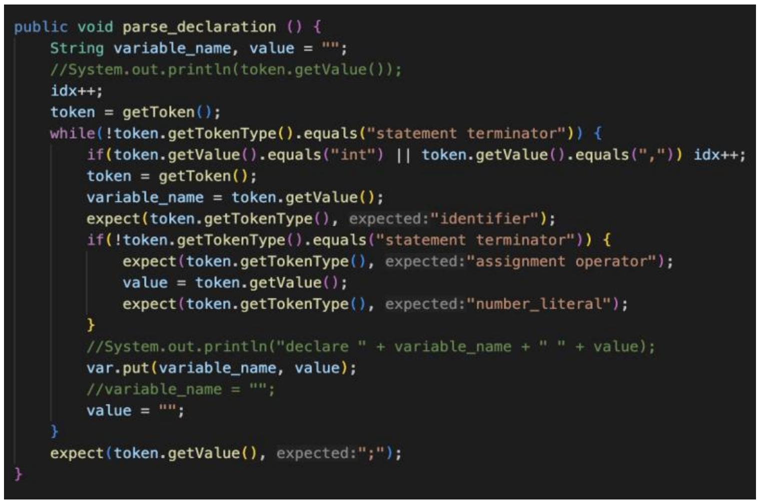
```java
public void start_parser() {
    idx = 0;
    token = getToken();
    int count = 0;

    parse_init();

    while (!token.getValue().equals("end")) { //while stack not empty
        //System.out.println(count++);
        //System.out.println("start begin");
        if(idx >= max_len) break;
        token = getToken();

        if (token.getValue().equals("int")) {
            //System.out.println("check " + token.getValue());
            parse_declaration();
        }
        else if(token.getTokenType().equals("identifier")) {
            parse_assignment();
        }
        else if (token.getValue().equals("print_line")) {
            parse_print();
        }
        else error(token.getValue());
    }

    parse_end();
    System.exit(status:0);
}
```

At BEGIN_BLOCK



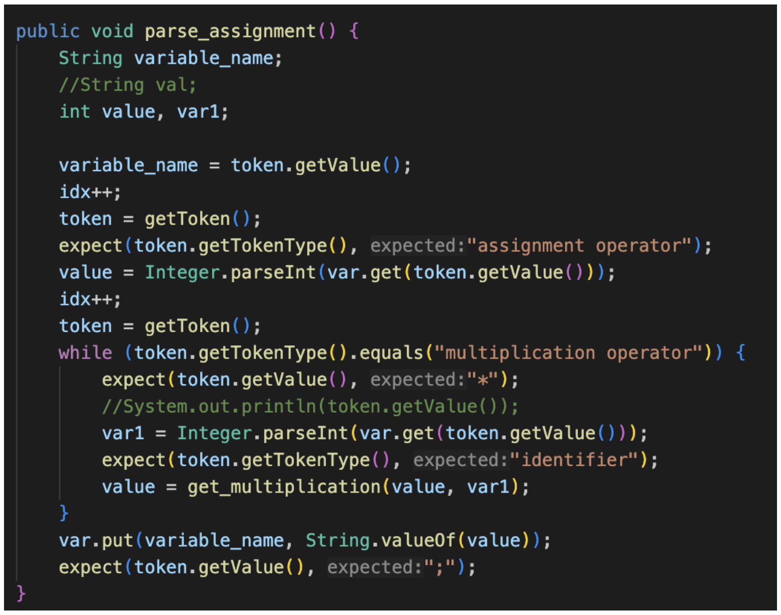
```java
public void parse_init () {
    //System.out.println(token.getValue());
    expect(token.getValue(), expected:"program");
    //System.out.println(token.getValue());
    expect(token.getTokenType(), expected:"identifier");
    //System.out.println(token.getValue());
    expect(token.getValue(), expected:"begin");
}
```

At DECLARATION, adding inherited values to symbol table



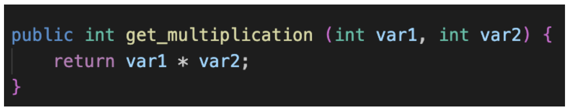
```java
public void parse_declaration () {
    String variable_name, value = "";
    //System.out.println(token.getValue());
    idx++;
    token = getToken();
    while(!token.getTokenType().equals("statement terminator")) {
        if(token.getValue().equals("int") || token.getValue().equals(",")) idx++;
        token = getToken();
        variable_name = token.getValue();
        expect(token.getTokenType(), expected:"identifier");
        if(!token.getTokenType().equals("statement terminator")) {
            expect(token.getTokenType(), expected:"assignment operator");
            value = token.getValue();
            expect(token.getTokenType(), expected:"number_literal");
        }
        //System.out.println("declare " + variable_name + " " + value);
        var.put(variable_name, value);
        //variable_name = "";
        value = "";
    }
    expect(token.getValue(), expected:";");
}
```

AT OP_STMT, getting and updating inherited values in symbol table



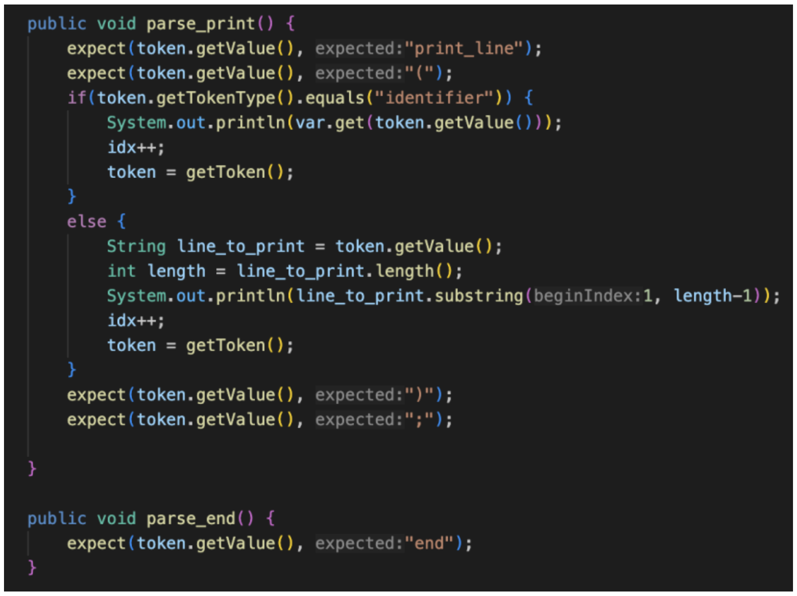
```java
public void parse_assignment() {
    String variable_name;
    //String val;
    int value, var1;

    variable_name = token.getValue();
    idx++;
    token = getToken();
    expect(token.getTokenType(), expected:"assignment operator");
    value = Integer.parseInt(var.get(token.getValue()));
    idx++;
    token = getToken();
    while (token.getTokenType().equals("multiplication operator")) {
        expect(token.getValue(), expected:"*");
        //System.out.println(token.getValue());
        var1 = Integer.parseInt(var.get(token.getValue()));
        expect(token.getTokenType(), expected:"identifier");
        value = get_multiplication(value, var1);
    }
    var.put(variable_name, String.valueOf(value));
    expect(token.getValue(), expected:";");
}
```

Logic for calculating value from inheritance



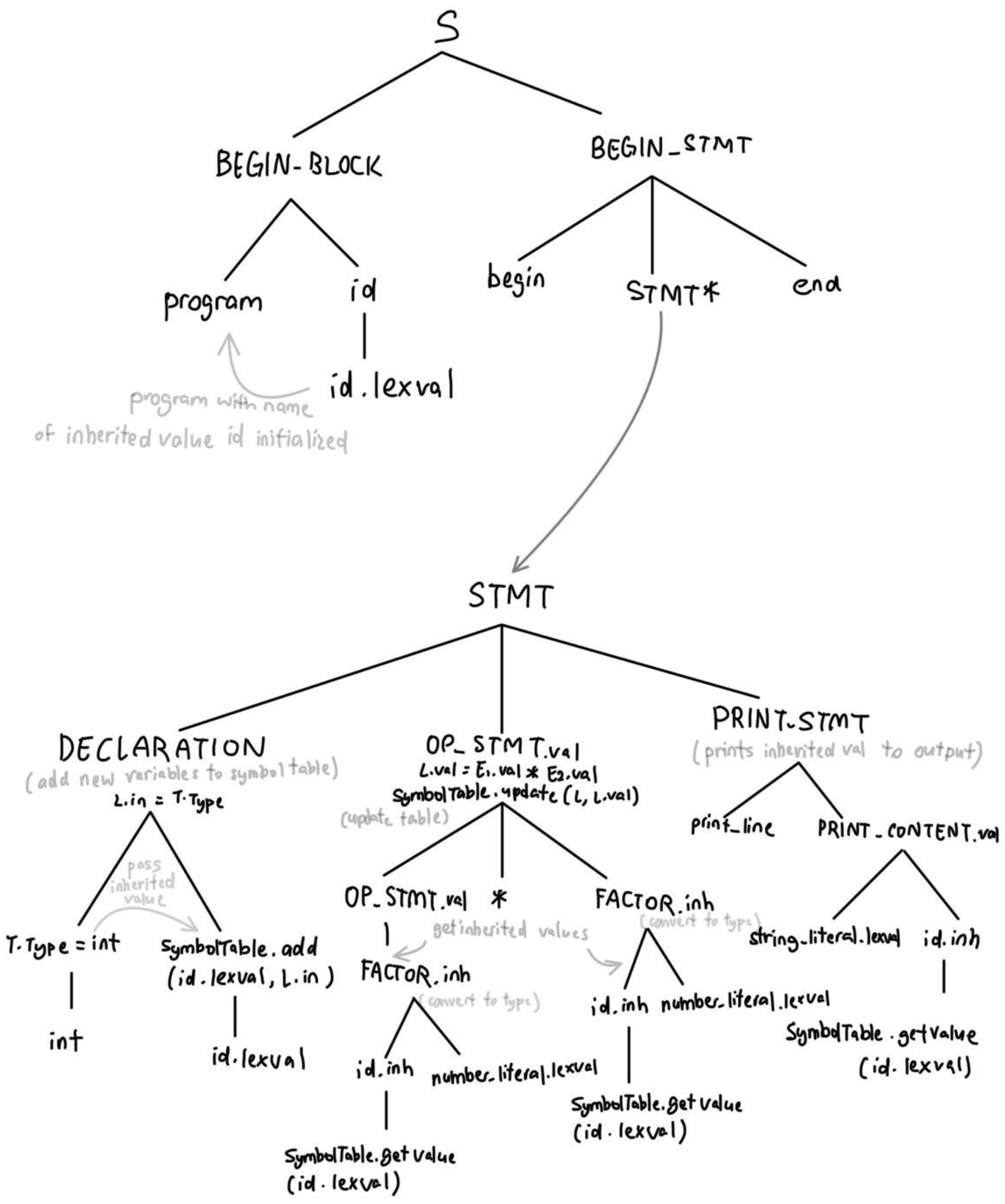
```java
public int get_multiplication (int var1, int var2) {
    return var1 * var2;
}
```

AT PRINT_STMT, printing output of relevant values in symbol table



```java
public void parse_print() {
    expect(token.getValue(), expected:"print_line");
    expect(token.getValue(), expected:"(");
    if(token.getTokenType().equals("identifier")) {
        System.out.println(var.get(token.getValue()));
        idx++;
        token = getToken();
    }
    else {
        String line_to_print = token.getValue();
        int length = line_to_print.length();
        System.out.println(line_to_print.substring(beginIndex:1, length-1));
        idx++;
        token = getToken();
    }
    expect(token.getValue(), expected:")");
    expect(token.getValue(), expected:";");

}

public void parse_end() {
    expect(token.getValue(), expected:"end");
}
```

Simple Flowchart


S
BEGIN-BLOCK
BEGIN_STMT
program
id
begin
STMT*
end
id.lexval
program with name
of inherited value id initialized
STMT
DECLARATION
(add new variables to symbol table)
L.in = T.Type
pass inherited value
T.Type = int
int
SymbolTable.add
(id.lexval, L.in)
id.lexval
OP_STMT.val
L.val = E1.val * E2.val
SymbolTable.update (L, L.val)
(update table)
OP_STMT.val
*
FACTOR.inh
get inherited values
FACTOR.inh
(convert to type)
id.inh
number_literal.lexval
SymbolTable.get Value
(id.lexval)
(convert to type)
id.inh
number_literal.lexval
SymbolTable.get Value
(id.lexval)
PRINT_STMT
(prints inherited val to output)
print_line
PRINT_CONTENT.val
string_literal.lexval
id.inh
SymbolTable.getValue
(id.lexval)