Corrected Grammar, FIRST and FOLLOW sets

Group 21

Honnesh Rohmetra	- 2016B2A7770P
S Hariharan	- 2017A7PS0134P
Praveen Ravirathinam	- 2017A7PS1174P
Anirudh Chakravarthy	- 2017A7PS1195P

Assumptions

1. Arithmetic operators (+, -, *, /) are left associative.
2. Boolean operators (AND, OR) are left associative
3. There can exist only one relational operator in arithmetic expressions, because the output of a relational operation would be boolean, which shouldn’t be compared against numeric values.

Modifications

1. Resolved left recursion in input parameter list. Resolved as follows:

```
<input_plist>     ->  ID COLON <dataType> <ipList’>
<ipList’>         ->  COMMA ID COLON <dataType> <ipList’> | ε
```

2. Resolved left recursion in output parameter list. Resolved as follows:

```
<output_plist>    ->  ID COLON <type> <opList’>
<opList’>         ->  COMMA ID COLON <type> <opList’> | ε
```

3. Expanded PRINT to accept TRUE and FALSE tokens. Done as follows:

```
<ioStmt>    -> GET_VALUE BO ID BC SEMICOL | PRINT BO <ioVar> BC SEMICOL
<ioVar>     -> <var> | <atoms>
```

4. Enabled array indexing and declaration using integer constants in addition to variables. Implemented as follows:

```
<whichId>   -> SQBO <index> SQBC |  ε
<dataType>  -> INTEGER|REAL|BOOLEAN| ARRAY SQBO <range> SQBC OF <type>
<range>     -> <index> RANGEOP <index>
```

5. Introduced a new non-terminal to resolve left recursion in non-terminal <idList>. Proceeded as follows:

```
<idList>    ->  ID <idList’>
<idList’>   ->  COMMA ID <idList’> | ε
```

6. Introduced precedence in arithmetic operators, and resolved subsequent left-recursion with an assumed left-associativity. Implemented as:

```
<arithmeticExpr>        -> <mulExpr> <arithmeticExpr’>
<arithmeticExpr’>       -> <addOp> <mulExpr> <arithmeticExpr’> | ε
<addOp>                 -> PLUS | MINUS
<mulExpr>               -> <factor> <mulExpr’>
<mulExpr’>              -> <mulOp> <factor> <mulExpr’> | ε
<mulOp>                 -> MUL | DIV
```

7. Implemented unary operators, and resolved left-factoring.

```
<expression>      ->  <abExpr> <expr'> | <U>
<U>               ->  MINUS <unaryExpr> | PLUS <unaryExpr>
<unaryExpr>       ->  <var> | BO <arithmeticExpr> BC
```

8. Made grammar LL(1) compatible by removing ambiguity between arithmetic and boolean logic expressions.

```
<expression>      -> <abExpr> <expr'> | <U>
<expr'>           -> <logicalOp> <abExpr> <expr'> | ε
<abExpr>          -> <arithmeticExpr> <relTerm>
<relTerm>         -> <relationalOp> <arithmeticExpr> | ε
```

9. Introduced left associativity in boolean operators, and resolved left-recursion.

```
<expression>      -> <abExpr> <expr'>
<expr'>           -> <logicalOp> <abExpr> <expr'> | ε
<logicalOp>       -> AND | OR
```

10. Enforced condition for at least 1 case statement, and resolved left factoring. Implemented as:

```
<conditionalStmt> -> SWITCH BO ID BC START <caseStmts> <default> END
<caseStmts>       -> <caseStmt> <multiCase>
<multiCase>       -> <caseStmt> <multiCase> | ε
<caseStmt>        -> CASE <value> COLON <statements> BREAK SEMICOL
```

11. Loop iteration variable in ‘for’ restricted to static constant integers. Done as follows:

```
<iterativeStmt>   -> FOR BO ID IN <loopRange> BC START <statements> END
<loopRange>       -> NUM RANGEOP NUM
```

MODIFIED GRAMMAR

1. <program> -> <moduleDeclarations> <otherModules> <driverModule> <otherModules>
2. <moduleDeclarations> -> <moduleDeclaration> <moduleDeclarations> | **ε**
3. <moduleDeclaration> -> **DECLARE MODULE ID** SEMICOL
4. <otherModules> -> <module> <otherModules>| **ε**
5. <driverModule> -> **DRIVERDEF DRIVER PROGRAM DRIVERENDDEF** <moduleDef>
6. <module> -> **DEF MODULE ID ENDDEF TAKES INPUT SQBO** <input_plist> **SQBC SEMICOL** <ret> <moduleDef>
7. <ret> -> **RETURNS SQBO** <output_plist> **SQBC SEMICOL | ε**
8. <input_plist> -> **ID COLON** <dataType> <ipList'>
9. <ipList'> -> **COMMA ID COLON** <dataType> <ipList'> | **ε**
10. <output_plist> -> **ID COLON** <type> <opList'>
11. <opList'> -> **COMMA ID COLON** <type> <opList'> | **ε**
12. <dataType> -> **INTEGER | REAL | BOOLEAN | ARRAY SQBO** <range> **SQBC OF** <type>
13. <type> -> **INTEGER | REAL | BOOLEAN**
14. <moduleDef> -> **START** <statements> **END**
15. <statements> -> <statement> <statements> | **ε**
16. <statement> -> <ioStmt> | <simpleStmt> | <declareStmt> | <conditionalStmt> | <iterativeStmt>
17. <ioStmt> -> **GET_VALUE BO ID BC SEMICOL | PRINT BO** <ioVar> **BC SEMICOL**
18. <ioVar> -> <var> | <atoms>
19. <var> -> **ID** <whichId> | **NUM | RNUM**
20. <whichId> -> **SQBO** <index> **SQBC** | **ε**
21. <simpleStmt> -> <assignmentStmt> | <moduleReuseStmt>
22. <assignmentStmt> -> **ID** <whichStmt>
23. <whichStmt> -> <lvalueIDStmt> | <lvalueARRStmt>
24. <lvalueIDStmt> -> **ASSIGNOP** <expression> **SEMICOL**
25. <lvalueARRStmt> -> **SQBO** <index> **SQBC ASSIGNOP** <expression> **SEMICOL**
26. <index> -> **NUM | ID**
27. <moduleReuseStmt> -> <optional> **USE MODULE ID WITH PARAMETERS** <idList> **SEMICOL**
28. <optional> -> **SQBO** <idList> **SQBC ASSIGNOP** | **ε**
29. <idList> -> **ID** <idList'>
30. <idList'> -> **COMMA ID** <idList'> | **ε**
31. <expression> -> <abExpr> <expr'> | <U>
32. <U> -> MINUS <unaryExpr> | PLUS <unaryExpr>
33. <unaryExpr> -> <var> | BO <arithmeticExpr> BC
34. <expr'> -> <logicalOp> <abExpr> <expr'> | **ε**
35. <abExpr> -> <arithmeticExpr> <relTerm>

36. <relTerm> -> <relationalOp> <arithmeticExpr> | **ε**

37. <arithmeticExpr> -> <mulExpr> <arithmeticExpr’>

38. <arithmeticExpr’> -> <addOp> <mulExpr> <arithmeticExpr’> | **ε**

39. <addOp> -> **PLUS | MINUS**

40. <mulExpr> -> <factor> <mulExpr’>

41. <mulExpr’> -> <mulOp> <factor> <mulExpr’> | **ε**

42. <mulOp> -> **MUL | DIV**

43. <factor> -> **BO** <expression> **BC** | <var> | <atoms>

44. <atoms> -> **TRUE | FALSE**

45. <logicalOp> -> **AND | OR**

46. <relationalOp> -> **LT | LE | GT | GE | EQ | NE**

47. <declareStmt> -> **DECLARE** <idList> **COLON** <dataType> **SEMICOL**

48. <conditionalStmt> -> **SWITCH BO ID BC START** <caseStmts><default> **END**

49. <caseStmts> -> <caseStmt> <multiCase>

50. <multiCase> -> <caseStmt> <multiCase> | **ε**

51. <caseStmt> -> **CASE** <value> **COLON** <statements> **BREAK SEMICOL**

52. <value> -> **NUM | TRUE | FALSE**

53. <default> -> **DEFAULT COLON** <statements> **BREAK SEMICOL | ε**

54. <iterativeStmt> -> **FOR BO ID IN** <loopRange> **BC START** <statements> **END |**
WHILE BO <expression> **BC START** <statements> **END**

55. <loopRange> -> **NUM RANGEOP NUM**

56. <range> -> <index> **RANGEOP** <index>

FIRST AND FOLLOW SETS

Non-terminal	First set	Follow set
<program>	DECLARE, DEF, DRIVERDEF	$
<moduleDeclarations>	DECLARE, ε	DEF, DRIVERDEF
<moduleDeclaration>	DECLARE	DECLARE, DEF, DRIVERDEF
<otherModules>	DEF, ε	DRIVERDEF, $
<driverModule>	DRIVERDEF	DEF, $
<module>	DEF	DEF, DRIVERDEF, $
<ret>	RETURNS, ε	START
<input_plist>	ID	SQBC
<ipList’>	COMMA, ε	SQBC
<output_plist>	ID	SQBC
<opList’>	COMMA, ε	SQBC
<dataType>	INTEGER, REAL, BOOLEAN, ARRAY	COMMA, SQBC, SEMICOL
<type>	INTEGER, REAL, BOOLEAN	COMMA. SQBC, SEMICOL
<moduleDef>	START	DEF, DRIVERDEF, $
<statements>	GET_VALUE, PRINT, ID, SQBO, ε, DECLARE, SWITCH, FOR, WHILE, USE	END, BREAK
<statement>	GET_VALUE, PRINT, ID, SQBO, DECLARE, SWITCH, FOR, WHILE, USE	GET_VALUE, PRINT, DECLARE, SWITCH, FOR, WHILE, ID, SQBO, USE, END, BREAK
<ioStmt>	GET_VALUE, PRINT	GET_VALUE, PRINT, DECLARE, SWITCH, FOR, WHILE, ID, SQBO, USE, END, BREAK
<ioVar>	ID, NUM, RNUM, TRUE, FALSE	BC
<var>	ID, NUM, RNUM	BC, MUL, DIV, PLUS, MINUS, SEMICOL, LT, LE, GT, GE, EQ, NE, AND, OR
<whichId>	SQBO, ε	BC, MUL, DIV, PLUS, MINUS,

		SEMICOL, LT, LE, GT, GE, EQ, NE, AND, OR
<simpleStmt>	ID, SQBO, USE	GET_VALUE, PRINT, DECLARE, SWITCH, FOR, WHILE, ID, SQBO, USE, END, BREAK
<assignmentStmt>	ID	GET_VALUE, PRINT, DECLARE, SWITCH, FOR, WHILE, ID, SQBO, USE, END, BREAK
<whichStmt>	ASSIGNOP, SQBO	GET_VALUE, PRINT, DECLARE, SWITCH, FOR, WHILE, ID, SQBO, USE, END, BREAK
<lvalueIDStmt>	ASSIGNOP	GET_VALUE, PRINT, DECLARE, SWITCH, FOR, WHILE, ID, SQBO, USE, END, BREAK
<lvalueARRStmt>	SQBO	GET_VALUE, PRINT, DECLARE, SWITCH, FOR, WHILE, ID, SQBO, USE, END, BREAK
<index>	NUM, ID	SQBC, RANGEOP
<moduleReuseStmt>	SQBO, USE	GET_VALUE, PRINT, DECLARE, SWITCH, FOR, WHILE, ID, SQBO, USE, END, BREAK
<optional>	SQBO, ε	USE
<idList>	ID	SEMICOL, SQBC, COLON
<idList’>	COMMA, ε	SEMICOL, SQBC, COLON
<expression>	BO, ID, NUM, RNUM, TRUE, FALSE, PLUS, MINUS	SEMICOL, BC
<U>	MINUS, PLUS	SEMICOL, BC
<unaryExpr>	ID, NUM, RNUM, BO	SEMICOL, BC
<expr’>	AND, OR, ε	SEMICOL, BC
<abExpr>	BO, ID, NUM, RNUM, TRUE, FALSE	AND, OR, SEMICOL, BC
<relTerm>	LT, LE, GT, GE, EQ, NE, ε	AND, OR, SEMICOL, BC
<arithmeticExpr>	BO, ID, NUM, RNUM, TRUE, FALSE	SEMICOL, BC, LT, LE, GT, GE, EQ, NE, AND, OR
<arithmeticExpr’>	PLUS, MINUS, ε	SEMICOL, BC, LT, LE, GT, GE, EQ, NE,

		AND, OR
<addOp>	PLUS, MINUS	BO, ID, NUM, RNUM, TRUE, FALSE
<mulExpr>	BO, ID, NUM, RNUM, TRUE, FALSE	PLUS, MINUS, SEMICOL, BC, LT, LE, GT, GE, EQ, NE, AND, OR
<mulExpr’>	MUL, DIV, ε	PLUS, MINUS, SEMICOL, BC, LT, LE, GT, GE, EQ, NE, AND, OR
<mulOp>	MUL, DIV	BO, ID, NUM, RNUM, TRUE, FALSE
<factor>	BO, ID, NUM, RNUM, TRUE, FALSE	MUL, DIV, PLUS, MINUS, SEMICOL, BC, LT, LE, GT, GE, EQ, NE, AND, OR
<atoms>	TRUE, FALSE	BC, MUL, DIV, PLUS, MINUS, SEMICOL, LT, LE, GT, GE, EQ, NE, AND, OR
<logicalOp>	AND, OR	ID, TRUE, FALSE, BO, NUM, RNUM
<relationalOp>	LT, LE, GT, GE, EQ, NE	ID, TRUE, FALSE, BO, NUM, RNUM
<declareStmt>	DECLARE	GET_VALUE, PRINT, DECLARE, SWITCH, FOR, WHILE, ID, SQBO, USE, END, BREAK
<conditionalStmt>	SWITCH	GET_VALUE, PRINT, DECLARE, SWITCH, FOR, WHILE, ID, SQBO, USE, END, BREAK
<caseStmts>	CASE	DEFAULT, END
<caseStmt>	CASE	CASE, DEFAULT, END
<multiCase>	CASE, ε	DEFAULT, END
<value>	NUM, TRUE, FALSE	COLON
<default>	DEFAULT, ε	END
<iterativeStmt>	FOR, WHILE	GET_VALUE, PRINT, DECLARE, SWITCH, FOR, WHILE, ID, SQBO, USE, END, BREAK
<range>	NUM, ID	SQBC
<loopRange>	NUM	BC