GROUP NO :30
ID: 2017A7PS0050P NAME : JIVAT NEET KAUR
ID: 2017A7PS0088P NAME : VAISHNAVI KOTTURU
ID: 2017A7PS0104P NAME : SARGUN SINGH
ID: 2017A7PS0160P NAME : ADITI MANDLOI
ID: 2017A7PS0227P NAME : JASPREET SINGH

S. No.	Grammar Rule	Abstract Syntax Tree Formation Rules
1	<program> → <moduleDeclarations>$_1$ <otherModules>$_1$ <driverModule> <otherModules>$_2$	<program>.addr = new Node ("program", <moduleDeclarations>.addr, <otherModules>$_1$.addr, <driverModule>.addr, <otherModules>$_2$.addr)
2	<moduleDeclarations>$_1$ → <moduleDeclaration> <moduleDeclarations>$_2$	<moduleDeclarations>$_1$.addr = new Node ("moduleDeclarations", <moduleDeclaration>.addr, <moduleDeclarations>$_2$.addr)
3	<moduleDeclarations> → e	<moduleDeclarations>.addr = NULL
4	<moduleDeclaration> → DECLARE MODULE ID SEMICOL	ID.type = MODULE <moduleDeclaration>.addr = new Leaf(ID, ID.entry)
5	<otherModules>$_1$ → <module> <otherModules>$_2$	<otherModules>$_1$.addr = new Node("otherModules", <module>.addr, <otherModules>$_2$.addr)
6	<otherModules> →e	<otherModules>.addr = NULL
7	<driverModule> → DRIVERDEF DRIVER PROGRAM DRIVERENDDEF <moduleDef>	<driverModule>.addr = new Node("driverModule", <moduleDef>.addr)

8	<module> → DEF MODULE ID ENDDEF TAKES INPUT SQBO <input_plist> SQBC SEMICOL <ret> <moduleDef>	ID.type = MODULE
		ID.addr = new Leaf(ID, ID.entry)
		<module>.addr = new Node("module", ID.addr, <input_plist>.head, <ret>.head, <moduleDef>.addr)

9	<ret> → RETURNS SQBO <output_plist> SQBC SEMICOL	<ret>.addr = new Node("ret", <output_plist>.addr)
10	<ret> → e	<ret>.addr = NULL

11	<input_plist> → ID COLON <dataType> <N1>	ID.addr = new Leaf(ID, ID.entry)
		temp = new Node ("input_plist_single", ID.addr, <dataType>.addr, NULL)
		<input_plist>.syn_list = insertAtBegin(<N1>.syn_list, temp)

12	<N1>$_1$ → COMMA ID COLON <dataType> <N1>$_2$	ID.addr = new Leaf(ID, ID.entry)
		temp = new Node ("input_plist_single", ID.addr, <dataType>.addr, NULL)
		<N1>1.syn_list = insertAtBegin(<N1>2.syn_list, temp)

13	<N1> → e	<N1>.syn_list = NULL

14	<output_plist> → ID COLON <type> <N2>	ID.addr = new Leaf(ID, ID.entry)
		temp = new Node ("output_plist_single", ID.addr, <type>.addr, NULL)
		<output_plist>.syn_list = insertAtBegin(<N2>.syn_list,temp)

15	<N2>$_1$ → COMMA ID COLON <type> <N2>$_2$	ID.addr = new Leaf(ID, ID.entry)
		temp = new Node ("output_plist_single", ID.addr, <type>.addr, NULL)
		<N2>1.syn_list = insertAtBegin(<N2>2.syn_list, temp)

16	<N2> → e	<N2>.syn_list = NULL

17	<dataType> → ARRAY SQBO <range_arrays> SQBC OF <type>	<dataType>.addr = new Node("dataType_array", <range_arrays>.addr, <type>.addr)
18	<dataType> → INTEGER	<dataType>.type = Integer
19	<dataType> → REAL	<dataType>.type = Real
20	<dataType> → BOOLEAN	<dataType>.type = Boolean
21	<range_arrays> → <index>$_1$ RANGEOP <index>$_2$	<range_arrays>.addr = new Node("range_arrays", <index>1.addr, <index>2.addr)
22	<type> → INTEGER	<type>.type = Integer
23	<type> → INTEGER	<type>.type = Real
24	<type> → BOOLEAN	<type>.type = Boolean
25	<moduleDef> → START <statements> END	<moduleDef>.addr = <statements>.addr Free (<statements>)
26	<statements>$_1$ → <statement> <statements>$_2$	<statements>$_1$.addr = new Node("statements", <statement>.addr, <statements>$_2$.addr)
27	<statements> →e	<statements>.addr = NULL
28	<statement> → <ioStmt>	<statement>.addr = <ioStmt>.addr Free (<ioStmt>)
29	<statement> → <simpleStmt>	<statement>.addr = <simpleStmt>.addr Free (<simpleStmt>)
30	<statement> → <declareStmt>	<statement>.addr = <declareStmt>.addr Free (<declareStmt>)
31	<statement> → <conditionalStmt>	<statement>.addr = <conditionalStmt>.addr Free (<conditionalStmt>)
32	<statement> → <iterativeStmt>	<statement>.addr = <iterativeStmt>.addr Free (<iterativeStmt>)
33	<ioStmt> → GET_VALUE BO ID BC SEMICOL	ID.addr = new Leaf(ID, ID.entry) <ioStmt>.addr= new Node("ioStmt", ID.addr)
34	<ioStmt> → PRINT BO <var> BC SEMICOL	<ioStmt>.addr = new Node("ioStmt",<var>.addr)
35	<var> → <var_id_num>	<var>.addr = <var_id_num>.addr Free (<var_id_num>)
36	<var> → <boolConstt>	<var>.addr = <boolConstt>.addr Free (<boolConstt>)
37	<boolConstt> → TRUE	<boolConstt>.addr = new Leaf(TRUE, 'true')
38	<boolConstt> → FALSE	<boolConstt>.addr = new Leaf(FALSE, 'false')
39	<var_id_num> → ID <whichId>	ID.addr = new Leaf(ID, ID.entry) <var_id_num>.addr = new Node("var_id_num", ID.addr, <whichId>.addr)
40	<var_id_num> → NUM	<var_id_num>.addr = new Leaf(NUM, NUM.value)
41	<var_id_num> → RNUM	<var_id_num>.addr = new Leaf(RNUM, RNUM.value)
42	<whichId> → SQBO <index> SQBC	<whichId>.addr = <index>.addr Free (<index>)

43	<whichId> → e	<whichId>.addr = NULL
44	<simpleStmt> → <assignmentStmt>	<simpleStmt> .addr = <assignmentStmt>.addr Free (<assignmentStmt>)
45	<simpleStmt> → <moduleReuseStmt>	<simpleStmt>.addr = <moduleReuseStmt>.addr Free (<moduleReuseStmt>)
46	<assignmentStmt> → ID <whichStmt>	ID.addr = new Leaf(ID, ID.entry) <assignmentStmt>.addr = new Node("assignmentStmt", ID.addr, <whichStmt>.addr)
47	<whichStmt> → <lvalueIDStmt>	<whichStmt>.addr = <lvalueIDStmt>.addr Free (<lvalueIDStmt>)
48	<whichStmt> → <lvalueARRStmt>	<whichStmt>.addr = <lvalueARRStmt>.addr Free (<lvalueARRStmt>)
49	<lvalueIDStmt> → ASSIGNOP <expression> SEMICOL	<lvalueIDStmt>.addr= <expression>.addr Free (<expression>)
50	<lvalueARRStmt> → SQBO <index> SQBC ASSIGNOP <expression> SEMICOL	<lvalueARRStmt>.addr = new Node("lvalueARRStmt", <index>.addr, <expression>.addr)
51	<index> → NUM	<index>.addr = new Leaf(NUM, NUM.value)
52	<index> → ID	<index>.addr = new Leaf(ID, ID.entry)
53	<moduleReuseStmt> → <optional> USE MODULE ID WITH PARAMETERS <idList> SEMICOL	ID.addr = new Leaf(ID, ID.entry) <moduleReuseStmt>.addr = new Node("moduleReuseStmt", <optional>.addr, ID.addr, <idList>.head)
54	<optional> → SQBO <idList> SQBC ASSIGNOP	<optional>.addr = <idList>.addr Free (<idList>)
55	<optional> →e	<optional>.addr = NULL
56	<idList> → ID <N3>	ID.addr = new Leaf(ID, ID.entry) <idList>.syn_list = insertAtBegin(<N3>.syn_list, ID.addr)
57	<N3>$_1$ → COMMA ID <N3>$_2$	ID.addr = new Leaf(ID, ID.entry) <N3>$_1$.syn_list = insertAtBegin(<N3>$_2$.syn_list, ID.addr)
58	<N3> → e	<N3>.addr = NULL
59	<expression> → <arithmeticOrBooleanExpr>	<expression>.addr = <arithmeticOrBooleanExpr>.addr Free (<arithmeticOrBooleanExpr>)
60	<expression> → <U>	<expression>.addr = <U>.addr Free (<U>)
61	<U> ⟶ <unary_op> <new_NT>	<U>.addr = new Node("U", <unary_op>.addr, <new_NT>.addr)
62	<new_NT> → BO <arithmeticExpr> BC	<new_NT>.addr = <arithmeticExpr>.addr Free (<arithmeticExpr>)
63	<new_NT> → <var_id_num>	<new_NT>.addr = <var_id_num>.addr Free (<var_id_num>)
64	<unary_op> → PLUS	<unary_op>.addr = new Leaf(PLUS, '+')
65	<unary_op> → MINUS	<unary_op>.addr = new Leaf(MINUS, '-')

66	<arithmeticOrBooleanExpr> → <AnyTerm> <N7>	<N7>.inh = <AnyTerm>.addr Free(<AnyTerm>) <arithmeticOrBooleanExpr>.addr = <N7>.syn
67	<N7>$_1$ → <logicalOp> <AnyTerm> <N7>$_2$	<N7>$_2$.inh = new Node(“ <logicalOp>.addr->lexeme”, <N7>$_1$.inh, <AnyTerm>.addr) <N7>$_1$.syn = <N7>$_2$.syn
68	<N7> →e	<N7>.syn = <N7>.inh
69	<AnyTerm> → <arithmeticExpr> <N8>	<N8>.inh = <arithmeticExpr>.addr Free(<arithmeticExpr>) <AnyTerm>.addr = <N8>.syn
70	<AnyTerm> → <boolConstt>	<AnyTerm>.addr = <boolConstt>.addr Free (<boolConstt>)
71	<N8> → <relationalOp> <arithmeticExpr>	<N8>.syn = new Node(" <relationalOp>.addr->lexeme”, <N8> .inh, <arithmeticExpr>.addr)
72	<N8> →e	<N8>.syn = <N8>.inh
73	<arithmeticExpr> → <term> <N4>	<N4>.inh = <term>.addr Free(<term>) <arithmeticExpr>.addr = <N4>.syn
74	<N4>$_1$ → <op1> <term> <N4>$_2$	<N4>$_2$.inh = new Node(" <op1>.addr->lexeme”, <N4>$_1$.inh, <term> .addr) <N4>$_1$.syn = <N4>$_2$.syn
75	<N4> → e	<N4>.syn = <N4>.inh
76	<term> → <factor> <N5>	<N5>.inh = <factor>.addr Free(<factor>) <term>.addr = <N5>.syn
77	<N5>$_1$ → <op2> <factor> <N5>$_2$	<N5>$_2$.inh = new Node("<op2>.addr->lexeme”, <N5>$_1$.inh, <factor> .addr) <N5>$_1$.syn = <N5>$_2$.syn
78	<N5> → e	<N5>.syn = <N5>.inh
79	<factor> → BO <arithmeticOrBooleanExpr> BC	<factor>.addr = <arithmeticOrBooleanExpr>.addr Free (<arithmeticOrBooleanExpr>)
80	<factor> → <var_id_num>	<factor>.addr = <var_id_num>.addr Free (<var_id_num>)
81	<op1> → PLUS	<op1>.addr = new Leaf(PLUS, '+')
82	<op1> → MINUS	<op1>.addr = new Leaf(MINUS, '-')
83	<op2> → MUL	<op2>.addr = new Leaf(MUL, '*')
84	<op2> → DIV	<op2>.addr = new Leaf(DIV, '/')
85	<logicalOp> → AND	<logicalOp>.addr = new Leaf(AND, ‘AND’)
86	<logicalOp> → OR	<logicalOp>.addr = new Leaf(OR, ‘OR’)
87	<relationalOp> → LT	<relationalOp>.addr = new Leaf(LT, ‘<’)
88	<relationalOp> → LE	<relationalOp>.addr = new Leaf(LE ‘<’=)
89	<relationalOp> → GT	<relationalOp>.addr = new Leaf(GT, ‘.’)
90	<relationalOp> → GE	<relationalOp>.addr = new Leaf(GE, ‘>=’)

91	<relationalOp> → NE	<relationalOp>.addr = new Leaf(NE, ‘!=’)
92	<relationalOp> → EQ	<relationalOp>.addr = new Leaf(EQ ‘==’)
93	<declareStmt> → DECLARE <idList> COLON <dataType> SEMICOL	<declareStmt>.addr = new Node("declareStmt",<idList>.head, <dataType>.addr)
94	<conditionalStmt> → SWITCH BO ID BC START <caseStmts> <default> END	ID.addr = new Leaf(ID, ID.entry) <conditionalStmt>.addr = new Node("conditionalStmt", ID.addr, <caseStmts>.head,<default>.addr)
95	<caseStmts> → CASE <value> COLON <statements> BREAK SEMICOL <N9>	temp = new Node ("caseStmt_single", <value>.addr, <statements>.addr, NULL) <caseStmts>.syn_list= insertAtBegin(<N9>.syn_list, temp)
96	<N9>$_1$ → CASE <value> COLON <statements> BREAK SEMICOL <N9>$_2$	temp = new Node ("caseStmt_single", <value>.addr, <statements>.addr, NULL) <N9>1.syn_list= insertAtBegin(<N9>2.syn_list, temp)
97	<N9> → e	<N9>.addr = NULL
98	<value> → NUM	<value>.addr = new Leaf(NUM, NUM.value)
99	<value> → TRUE	<value>.addr = new Leaf(TRUE, 'true')
100	<value> → FALSE	<value>.addr = new Leaf(FALSE, 'false')
101	<default> → DEFAULT COLON <statements> BREAK SEMICOL	<default>.addr = <statements>.addr Free (<statements>)
102	<default> → e	<default>.addr = NULL
103	<iterativeStmt> → FOR BO ID IN <range> BC START <statements> END	ID.addr = new Leaf(ID, ID.entry) <iterativeStmt>.addr = new Node("iterativeStmt_For", ID.addr, <range>.addr, <statements>.addr)
104	<iterativeStmt> → WHILE BO <arithmeticOrBooleanExpr> BC START <statements> END	<iterativeStmt>.addr = new Node("iterativeStmt_While", <arithmeticOrBooleanExpr>.addr, <statements>.addr)

105	<range> → NUM$_1$ RANGEOP NUM$_2$	NUM$_1$.addr = new Leaf(NUM, NUM.value)
		NUM$_2$.addr = new Leaf(NUM, NUM.value)
		<range>.addr = new Node("range", NUM1.addr, NUM2.addr)