Group No. - 46

Name of Team Members:
Piklu Paul: 2017A7PS0006P
Rajababu Saikia: 2017A7PS0007P
Saurav Virmani: 2017A7PS0090P
Siddhant Kharbanda: 2017A7PS0111P
Sreyas Ravichandran: 2017A7PS0275P

Rule	AST Rules
1. <program> => <moduleDeclarations> <otherModules><driverModule><otherModules>	program.syn = makenode("PROGRAM",moduleDeclarations.syn, driverModule.syn, otherModules2.syn); otherModules1.inh = moduleDeclarations.syn; otherModules2.inh = otherModules1.syn
2. <moduleDeclarations> => <moduleDeclaration><moduleDeclarations>	moduleDeclarations2.inh = makenode("MODULE_DECLARATION",moduleDeclarations1.inh, moduleDeclaration.syn); moduleDeclarations1.syn = moduleDeclarations2.syn
3. <moduleDeclarations> => ε	moduleDeclarations.syn = moduleDeclarations.inh
4. <moduleDeclaration> => DECLARE MODULE ID SEMICOL	moduleDeclaration.syn = makenode("MODULE_DECLARATION", ID.entry)
5. <otherModules> => <module><otherModules>	otherModules1.syn = otherModules2.syn; otherModules2.inh = makenode(module.syn, otherModules1.inh); module.inh = otherModules1.inh
6. <otherModules> => ε	otherModules.syn = otherModules.inh
7. <driverModule> => DRIVERDEF DRIVER PROGRAM DRIVERENDDEF <moduleDef>	driverDef.syn = makenode("DRIVER_MODULE",moduleDef.syn); moduleDef.inh = driverModule.inh
8. <module> => DEF MODULE ID ENDDEF TAKES INPUT SQBO <input_plist> SQBC SEMICOL <ret><moduleDef>	module.syn = makenode("MODULE_DEFINITION", ID.entry, input_plist.syn, ret.syn, moduleDef.syn), moduleDef.inh=makenode("MODULE_DEF",module.inh,input_plist.syn, ret.syn, ID.entry)
9. <ret> => ε	ret.syn = ret.inh
10. <ret> => RETURNS SQBO <output_plist> SQBC SEMICOL	ret.syn = output_plist.syn
11. <input_plist> => ID COLON <dataType><N1>	input_plist.syn = N1.syn; N1.inh = append(NULL, {ID.entry, dataType.syn}); dataType.inh = input_plist.inh
12. <N1> => COMMA ID COLON <dataType> <N1>	N11.syn = N12.syn; N12.inh = append(N11.inh, {ID.entry, dataType.syn}); dataType.inh = N1.inh
13. <N1> => ε	N1.syn = makenode(N1.inh);
14. <output_plist> => ID COLON <dataType><N2>	output_plist.syn = N2.syn; N2.inh = append(NULL, {ID.entry, dataType.syn}); dataType.inh = output_plist.inh
15. <N2> => COMMA ID COLON <dataType> <N2>	N21.syn = N22.syn; N22.inh = append(N21.inh, {ID.entry, dataType.syn}); dataType.inh = N2.inh
16. <N2> => ε	N2.syn = makenode(N2.inh);
17. <dataType> => INTEGER	dataType.syn = makeLeaf("TYPE",INTEGER)
18. <dataType> => REAL	dataType.syn = makeLeaf("TYPE",REAL)
19. <dataType> => BOOLEAN	dataType.syn = makeLeaf("TYPE",BOOLEAN)
20. <dataType> => ARRAY SQBO <range_arrays> SQBC OF <type>	dataType.syn = array(range_arrays.syn, type.syn); range_arrays.inh = dataType.inh
21. <range_arrays> => <index> RANGEOP <index>	range_array.syn = makenode("RANGEOP",index1.syn, index2.syn); index1.inh = range_arrays.inh; index2.inh = range_arrays.inh
22. <type> => INTEGER	type.syn = makeLeaf("TYPE",INTEGER)
23. <type> => REAL	type.syn = makeLeaf("TYPE",REAL)
23. <type> => BOOLEAN	type.syn = makeLeaf("TYPE",BOOLEAN)
24. <moduleDef> => START <statements> END	moduleDef.syn = statements.syn; statements.inh = moduleDef.inh
25. <statements> =><statement> <statements>	statements1.syn = statements2.syn; statements2.inh = makenode("STATEMENTS",statement.syn, statements1.inh); statement.inh = statements1.inh
26. <statements> => ε	statements.syn = statements.inh
27. <statement> => <ioStmt>	statement.syn = ioStmt.syn ; ioStmt.inh = statement.inh
28. <statement> => <simpleStmt>	statement.syn = simpleStmt.syn; simpleStmt.inh = statement.inh
29. <statement> => <declareStmt>	statement.syn = declareStmt.syn; declareStmt.inh = statement.inh
30. <statement> => <condionalStmt>	statement.syn = conditionalStmt.syn; conditionalStmt.inh = statement.inh
31. <statement> => <iterativeStmt>	statement.syn = iterativeStmt.syn; iterativeStmt.inh = statement.inh
32. <ioStmt> => GET_VALUE BO ID BC SEMICOL	iostmt.syn = makenode("GET_VALUE", ID.entry)
33. <ioStmt> => PRINT BO <var> BC SEMICOL	iostmt.syn = makenode("PRINT_VALUE", var.syn); var.inh=ioStmt.inh
34. <var> => <boolConstt>	var.syn = boolConstt.syn
35. <var> => <var_id_num>	var.syn = var_id_num.syn; var_id_num.inh = var.inh
36. <boolConstt> => TRUE	boolConstt.syn = makeLeaf("BOOL",TRUE)
37. <boolConstt> => FALSE	boolConstt.syn = makeLeaf("BOOL",FALSE)
38. <var_id_num> => ID <whichId>	var_id_num.syn = makenode("VAR_ID",ID.entry, whichID.syn); whichID.inh = var_id_num.inh
39. <var_id_num> => NUM	var_id_num.syn = makeLeaf("NUM",NUM.val)
40. <var_id_num> => RNUM	var_id_num.syn = makeLeaf("RNUM",RNUM.val)
41. <whichId> => SQBO <index> SQBC	whichID.syn = index.syn; index.inh=whichID.inh
42. <whichId> => ε	whichID.syn = NULL
41. <simpleStmt> => <assignmentStmt>	simpleStmt.syn = assignmentStmt.syn, assignmentStmt.inh = simpleStmt.inh
44. <simpleStmt> => <moduleReuseStmt>	simpleStmt.syn = moduleReuseStmt.syn; moduleReuseStmt.inh = simpleStmt.inh
45. <assignmentStmt> => ID <whichStmt>	assignmentStmt.syn = whichStmt.syn; whichStmt.inh = ID.entry
46. <whichStmt> =><lvalueIDStmt>	whichStmt.syn = lvalueIDStmt.syn; lvalueIDstmt.inh = whichStmt.inh
47. <whichStmt> => <lvalueARRStmt>	whichStmt.syn = lvalueARRStmt.syn; lvalueARRstmt.inh = whichStmt.inh
48. <lvalueIDStmt> => ASSIGNOP <expression> SEMICOL	lvalueIDStmt.syn = makenode("EQUATE",lvalueIDStmt.inh, expression.syn); expression.inh = lvalueIDStmt.inh
49. <lvalueARRStmt> => SQBO <index> SQBC ASSIGNOP <expression> SEMICOL	lvalueARRStmt.syn = makenode("EQUATE",findArrPos(lvalueARRStmt.inh, index.syn), expression.syn); index.inh = lvalueARRStmt.inh; expression.inh = lvalueARRStmt.inh
50. <index> => NUM | ID	index.syn = NUM.val | index.syn = ID.entry
51. <moduleReuseStmt> =><optional> USE MODULE ID WITH PARAMETERS <idList>SEMICOL	moduleReuseStmt.syn = makenode("FUNC_CALL",optional.syn, ID.entry, idList.syn); optional.inh = moduleReuseStmt.inh; idList.inh = moduleReuseStmt.inh
52. <optional> => SQBO <idList> SQBC ASSIGNOP	optional.syn = makeNode("RETURN_VALS", idList.syn); idList.inh = optional.inh
54. <optional> => ε	optional.syn = NULL
55. <idList> => ID <N3>	idList.syn = N3.syn; N3.inh = append(NULL, {ID.entry})
56. <N3> => COMMA ID <N3>	N31.syn = N32.syn; N32.inh = append(N31.inh, {ID.entry})
57. <N3> => ε	N3.syn = makenode("ID_LIST", N3.inh);
58. <expression> => <arithmeticOrBooleanExpr>	expression.syn = arithmeticOrBooleanExpr.syn; arithmeticOrBooleanExpr.inh = expression.inh
58. <expression> => <U>	expression.syn = U.syn; U.inh = expression.inh
60. <U>=> <unary_op> <new_NT>	U.syn = makenode("UNARY_EXPR", unary_op.syn, new_NT.syn); new_NT.inh = U.inh
61. <new_NT> => BO <arithmeticExpr> BC	new_NT.syn = arithmeticExpr.syn, arithmenticExpr.inh = new_NT.inh
62. <new_NT> => <var_id_num>	new_NT.syn = var_id_num.syn; var_id_num.inh = new_NT.inh
63. <unary_op> => MINUS	unary_op.syn = makeNode("UOp", '-')
64. <unary_op> => PLUS	unary_op.syn = makeNode("UOp", '+')
68. <arithmeticOrBooleanExpr> => <AnyTerm> <N7>	arithmeticOrBooleanExpr.syn = N7.syn; N7.inh = makenode("A_OR_BOOL_EXPR",AnyTerm.syn, arithmeticOrBooleanExpr.inh); AnyTerm.inh = arithmeticOrBooleanExpr.inh
69. <N7> => ε	N7.syn = N7.inh
70. <N7> => <logicalOp> <AnyTerm> <N7>	N71.syn = N72.syn; N72.inh = makeNode("LOGICAL_EXPR",logicalOp.syn, N71.inh, AnyTerm.syn); Anyterm.inh =N71.inh
71. <AnyTerm> => <arithmeticExpr> <N8>	AnyTerm.syn = N8.syn; N8.inh = makenode("ARITH_EXPR",arithmeticExpr.syn, AnyTerm.inh); arithmeticExpr.inh = AnyTerm.inh
72. <AnyTerm> => <boolConstt>	AnyTerm.syn = boolConstt.syn;
73. <N8> => <relationalOp> <arithmeticExpr>	N8.syn = arithmeticExpr.syn; arithmeticExpr.inh = makeNode("RELATIONAL_EXPR", relationalOp.syn, N8.inh)
74. <N8> => ε	N8.syn = N8.inh
75. <arithmeticExpr> => <term> <N4>	arithmeticExpr.syn = N4.syn; N4.inh = makenode("TERM", term.syn, arithmeticExpr.inh)
76. <N4> => <op1> <term> <N4>	N41.syn = N42.syn; N42.inh = makeNode("TERM_EXPR", op1.syn, term.syn, N41.inh); term.inh = N41.inh
77. <N4> => ε	N4.syn = N4.inh
78. <term> => <factor> <N5>	term.syn = N5.syn; N5.inh = makenode("FACTOR", factor.syn, term.inh); factor.inh = term.inh
80. <N5> => <op2> <factor> <N5>	N51.syn = N52.syn; N52.inh = makenode("FACTOR_EXPR", op2.syn, factor.syn, N51.inh); factor.inh = N51.inh
81. <N5> => ε	N5.syn = N5.inh
82. <factor> => BO <arithmeticOrBooleanExpr> BC	factor.syn = arithmeticOrBooleanExpr.syn; arithmeticOrBoolean.inh = factor.inh
83. <factor> => <var_id_num>	factor.syn = var_id_num.syn
85. <op1> => MINUS	op1.syn = makeNode("ASOP",'-')
86. <op1> => PLUS	op1.syn = makeNode("ASOP",'+')
87. <op2> => MUL	op2.syn = makeNode("MDOP", '*')
88. <op2> => DIV	op2.syn = makeNode("MDOP", '/')
89. <logicalOp> => AND	logicalOp.syn = makeNode("LOGOP",AND)
90. <logicalOp> => OR	logicalOp.syn = makeNode("LOGOP",OR)
91. <relationalOp> => LT	relationalOp.syn = makeLeaf("RELOP",'<')
92. <relationalOp> => LE	relationalOp.syn = makeLeaf("RELOP",'<=')
93. <relationalOp> => GT	relationalOp.syn = makeLeaf("RELOP",'>')
94. <relationalOp> => GE	relationalOp.syn = makeLeaf("RELOP",'>=')
95. <relationalOp> => EQ	relationalOp.syn = makeLeaf("RELOP",'==')
96. <relationalOp> => NE	relationalOp.syn = makeLeaf("RELOP",'!=')
97. <declareStmt> => DECLARE <idList> COLON <dataType> SEMICOL	declareStmt.syn = makenode("DECLARE",dataType.syn, idList.syn); idList.inh = declareStmt.inh; dataType.inh = declareStmt.inh
98. <conditionalStmt> => SWITCH BO ID BC START <caseStmts> <default> END	conditionalStmt.syn = makenode("CONDITIONAL",caseStmts.syn, default.syn); caseStmts.inh = conditionalStmt.inh; default.inh = conditionalStmt.inh
99. <caseStmts> => CASE <value> COLON <statements> BREAK SEMICOL <N9>	caseStmts.syn = N9.syn; N9.inh = makenode("CASES",value.syn, statements.syn); statements.inh=N9.inh
100. <N9> => CASE <value> COLON <statements> BREAK SEMICOL <N9>	N91.syn = N92.syn; N92.inh = makenode("CASE",value.syn, statements.syn, N91.inh); statements.inh=N91.inh
101. <N9> => ε	N9.syn = N9.inh
102. <value> => NUM	value.syn = makeLeaf("NUM",NUM.val)
103. <value> => TRUE	value.syn = makeLeaf("BOOL",TRUE)
104. <value> => FALSE	value.syn = makeLeaf("BOOL",FALSE)
105. <default> => DEFAULT COLON <statements> BREAK SEMICOL	default.syn = makenode("DEFAULT", statements.syn); statements.inh= default.inh
106. <default> => ε	default.syn = NULL
107. <iterativeStmt> => FOR BO ID IN <range> BC START <statements> END	iterativeStmt.syn = makeNode("FOR", ID.entry, range.syn, statemets.syn); statements.inh= iterativeStmt.inh
108. <iterativeStmt> => WHILE BO <arithmeticOrBooleanExpr> BC START <statements> END	iterativeStmt.syn = makeNode("WHILE", arithmeticOrBooleanExpr.syn, statemets.syn); statements.inh= iterativeStmt.inh; arithmeticOrBooleanExpr.inh =iterativeStmt.inh;
109. <range> => NUM RANGEOP NUM	range.syn = makenode("RANGEOP",NUM1.val,NUM2.val)