grammar

1 <prog> - <modDecs> <othermods> <driver> <othermods>
2 <modDecs> - <modDec> <modDecs>
3 <modDecs> - eps
4 <modDec> - DECLARE MODULE ID SEMICOL
5 <othermods> - <mod> <othermods>
6 <othermods> - eps
7 <driver> - DRIVERDEF DRIVER PROGRAM DRIVERENDDEF <modDef>
8 <mod> - DEF MODULE ID ENDDEF TAKES INPUT SQBO <ip_list> SQBC SEMICOL <ret>
9 <modDef>
10 <ret> - RETURNS SQBO <op_list> SQBC SEMICOL
11 <ret> - eps
12 <ip_list> - ID COLON <dataType> <N1>
13 <N1> - COMMA ID COLON <dataType> <N1>
14 <N1> - eps
15 <op_list> - ID COLON <type> <N2>
16 <N2> - COMMA ID COLON <type> <N2>
17 <N2> - eps
18 <dataType> - INTEGER
19 <dataType> - REAL
20 <dataType> - BOOLEAN
21 <dataType> - ARRAY SQBO <range_arrays> SQBC OF <type>
22 <range_arrays> - <index_arr> RANGEOP <index_arr>
23 <type> - INTEGER
24 <type> - REAL
25 <type> - BOOLEAN
26 <modDef> - START <stmts> END
27 <stmts> - <stmt> <stmts>
28 <stmts> - eps
29 <stmt> - <iostmt>
30 <stmt> - <simplestmt>
31 <stmt> - <declarestmt>
32 <stmt> - <condstmt>
33 <stmt> - <iterstmt>
34 <iostmt> - GET_VALUE BO ID BC SEMICOL
35 <iostmt> - PRINT BO <var_print> BC SEMICOL
36 <boolconst> - TRUE
37 <boolconst> - FALSE
38 <id_num_rnum> - ID
39 <id_num_rnum> - NUM
40 <id_num_rnum> - RNUM
41 <arr_ele_print> - ID SQBO <index_arr> SQBC
42 <var_print> - ID <P1>
43 <var_print> - NUM
44 <var_print> - RNUM
45 <var_print> - <boolconst>
46 <P1> - SQBO <new_index> SQBC
47 <P1> - eps
48 <simplestmt> - <assignstmt>
49 <simplestmt> - <modReusestmt>
50 <assignstmt> - ID <whichstmt>
51 <whichstmt> - <lvalueIDstmt>
52 <whichstmt> - <lvalueARRstmt>
53 <lvalueIDstmt> - ASSIGNOP <exp> SEMICOL
54 <lvalueARRstmt> - SQBO <element_index_with_expressions> SQBC ASSIGNOP <exp> SEMICOL
55 <index_arr> - <sign> <new_index>
56 <new_index> - NUM
57 <new_index> - ID
58 <sign> - PLUS
59 <sign> - MINUS
60 <sign> - eps
61 <modReusestmt> - <optional> USE MODULE ID WITH PARAMETERS <paralist> SEMICOL
62 <paralist> - <sign> <actual_para_list> <paralist2>
63 <paralist2> - COMMA <sign> <actual_para_list> <paralist2>
64 <paralist2> - eps
65 <actual_para_list> - NUM
66 <actual_para_list> - RNUM
<actual_para_list> - <boolconst>

67 <actual_para_list> - ID <N_11>
68 <optional> - SQBO <idList> SQBC ASSIGNOP
69 <optional> - eps
70 <idList> - ID <N3>
71 <N3> - COMMA ID <N3>
72 <N3> - eps
73 <exp> - <arithOrboolExp>
74 <exp> - <U>
75 <U> - <unary_op> <new_NT>
76 <unary_op> - PLUS
77 <unary_op> - MINUS
78 <new_NT> - BO <arithExp> BC
79 <new_NT> - <id_num_rnum>
80 <arithOrboolExp> - <AnyTerm> <N4>
81 <N4> - <logOp> <AnyTerm> <N4>
82 <N4> - eps
83 <AnyTerm> - <arithExp> <N5>
84 <AnyTerm> - <boolconst>
85 <N5> - <relOp> <arithExp>
86 <N5> - eps
87 <arithExp> - <term> <N6>
88 <N6> - <op1> <term> <N6>
89 <N6> - eps
90 <term> - <factor> <N7>
91 <N7> - <op2> <factor> <N7>
92 <N7> - eps
93 <factor> - BO <arithOrboolExp> BC
94 <factor> - NUM
95 <factor> - RNUM
96 <factor> - ID <N_11>
97 <N_11> - SQBO <element_index_with_expressions> SQBC
98 <N_11> - eps
99 <arr_element> - ID SQBO <element_index_with_expressions> SQBC
100 <element_index_with_expressions> - <op1> <N_10>
101 <element_index_with_expressions> - <arrExpr>
102 <N_10> - <new_index>
103 <N_10> - BO <arrExpr> BC
104 <arrExpr> - <arrTerm> <arr_N4>
105 <arr_N4> - <op1> <arrTerm> <arr_N4>
106 <arr_N4> - eps
107 <arrTerm> - <arrFactor> <arr_N5>
108 <arr_N5> - <op2> <arrFactor> <arr_N5>
109 <arr_N5> - eps
110 <arrFactor> - ID
111 <arrFactor> - NUM
112 <arrFactor> - <boolconst>
113 <arrFactor> - BO <arrExpr> BC
114 <op1> - PLUS
115 <op1> - MINUS
116 <op2> - MUL
117 <op2> - DIV
118 <logOp> - AND
119 <logOp> - OR
120 <relOp> - LT
121 <relOp> - LE
122 <relOp> - GT
123 <relOp> - GE
124 <relOp> - EQ
125 <relOp> - NE
126 <declarestmt> - DECLARE <idList> COLON <dataType> SEMICOL
127 <condstmt> - SWITCH BO ID BC START <caseStmts> <default> END
128 <caseStmts> - CASE <value> COLON <stmts> BREAK SEMICOL <N9>
129 <N9> - CASE <value> COLON <stmts> BREAK SEMICOL <N9>
130 <N9> - eps
131 <value> - NUM
132 <value> - TRUE
133 <value> - FALSE
134 <default> - DEFAULT COLON <stmts> BREAK SEMICOL

135 <default> - eps
136 <iterstmt> - FOR BO ID IN <range_forloop> BC START <stmts> END
137 <iterstmt> - WHILE BO <arithOrboolExp> BC START <stmts> END
138 <range_forloop> - <index_forloop> RANGEOP <index_forloop>
139 <index_forloop> - <sign_forloop> <new_index_forloop>
140 <new_index_forloop> - NUM
141 <sign_forloop> - PLUS
142 <sign_forloop> - MINUS
143 <sign_forloop> - eps

GROUP-20

MEMBERS	ID : 2019B4A7
RAJAN	0572P
YASH	0638P
AYUSH	0652P
VASU	0656P
SUDARSHAN	0744P

SEMANTIC RULES FOR ABSTRACT SYNTAX TREE (AST) CREATION.

NOTE :- all rules traversal order : POST ORDER traversal.
(Left → Right → Root)

:- assume that *_syn attributes are first created (in post-order fashion).

<Rule no.> — {<Semantic Rules>}, Format

1. { < POST_ORDER > :
   <prog>.node_syn = make_node ("PROGRAM",
   make_node ("MODULEDECLARATIONS", <modDecs>.list_syn),
   make_node ("OTHERMODULES", <otherMods>.list_syn),
   make_node ("DRIVER", <driver>.node_syn),
   make_node ("OTHERMODULES1", <otherMods>$_1$.list_syn));
   free(<modDecs>); free(<otherMods>); free(<driver>); free(<otherMods>$_1$); }

2. { < POST_ORDER > :
   <modDecs>.list_syn = insert_at_head (<modDecs>$_1$.list_syn, <modDec>.node_syn);
   free(<modDec>); free(<modDecs>$_1$); }

3. { < POST_ORDER > :
   <modDecs>.list_syn = NULL
   free(eps); }

4. { < POST_ORDER > :
   <modDec>.node_syn = ID.addr;
   free(DECLARE); free(MODULE); free(SEMICOL); }

5. { < POST_ORDER > :
   <otherMods>.list_syn = insert_at_head (<otherMods>$_1$.list_syn,
   <mod>.node_syn);
   free(<mod>);
   free(<otherMods>$_1$); }

6. { <POST-ORDER> :

<otherMods>.list-syn = NULL;

free (eps);

}

7. { <POST-ORDER> :

<driver>.node-syn = <modDef>.node-syn

free (DRIVERDEF); free (DRIVER); free (PROGRAM); free (DRIVERENDDEF);

free (<modDef>); }

8. { <POST-ORDER> :

<mod>.node-syn = make_node (ID.name, <ip_list>.list-syn,

<ret>.list-syn, <modDef>.node-syn);

free (DEF); free (MODULE); free (ENDDEF); free (TAKES); free (INPUT);

free (SQBO); free (<ip-plist>); free (SQBC); free (SEMICOL); free (<ret>);

free (<modDef>); }

9. { <POST-ORDER> :

<ret>.list-syn = <op-plist>.list-syn ;

free (RETURNS); free (SQBO); free (SQBC); free (<op-plist>);

free (SEMICOL); }

10. { <POST-ORDER> :

<ret>.list-syn = NULL

free (eps); }

11. { <POST-ORDER> :

<ip-plist>.node-syn = make_node (ID.name, <dataType>.node-syn);

<ip-plist>.list-syn = insert_at_head (<N1>.list-syn, <ip-plist>.node-syn);

free (COLON); free (<dataType>); free (<N1>); }.

12. { <POST-ORDER> :

<N1>.node-syn = make_node (ID.name, <dataType>.node-syn);

<N1>.list-syn = insert_at_head (<N1>$_1$.list-syn, <N1>.node-syn);

free (COMMA); free (COLON); free (<dataType>); free (<N1>$_1$); }

13. { <N1>.list-syn = NULL; free(eps); }

14. { <op-list>.node-syn = make-node(ID.name, <type>.node-syn);
<op-list>.list-syn = insert_at_head(<N2>.list-syn, <op-list>.node-syn);
free(COLON); free(<type>); free(<N2>); }

15. { <N2>.node-syn = make-node(ID.name, <type>.node-syn);
<N2>.list-syn = insert-at-head(<N2>$_1$.list-syn, <N2>.node-syn);
free(COMMA); free(COLON); free(<type>); free(<N2>$_1$); }

16. { <N2>.list-syn = NULL;
free(eps); }.

17. { <dataType>.node-syn = INTEGER.addr; }

18. { <dataType>.node-syn = REAL.addr; }

19. { <dataType>.node-syn = BOOLEAN.addr; }

20. { <dataType>.node-syn = make-node("ARRAY", <range_arrays>.node-syn,
<type>.node-syn);
free(<range_arrays>);
free(<type>); free(ARRAY); free(SQBO); free(SQBC); free(OF); }

21. { <range-arrays>.node-syn = makenode("ARRAYRANGE", <index_array>$_1$.node-syn
, <index_array>$_2$.node-syn);
free(<index_array>$_1$);
free(<index_array>$_2$);
free(RANGEOP); }

22. { <type>.node-syn = INTEGER.addr; }

23. { <type>.node-syn = REAL.addr; }

24. { <type>.node-syn = BOOLEAN.addr; }

25. { <modDef>.node-syn = make.node("STATEMENTS", <stmt>.list-syn);
free(START); free(<stmts>); free(END); }

26. { <stmts>.list-syn = insert_at_head(<stmts>$_1$.list-syn, <stmt>.node-syn);
free(<stmt>); free(<stmts>); }

27. { <stmts>.list-syn = NULL; free(eps); }

28. { <stmt>.node-syn = <iostmt>.node-syn ;
free(<iostmt>); }

29. { <stmt>.node-syn = <simplestmt>.node_syn ; free(<simplestmt>); }

30. { <stmt>.node-syn = <declarestmt>.node-syn ; free(<declarestmt>); }

31. { <stmt>.node-syn = <condstmt>.node-syn ; free(<condstmt>); }

32. { <stmt>.node-syn = <iterstmt>.node-syn ; free(<iterstmt>); }

33. { <iostmt>.node-syn = makenode("INPUT", ID.addr);
free(GET_VALUE); free(BO); free(BC); free(SEMICOL); }

34. { <iostmt>.node-syn = <var_print>.node-syn ;
free(PRINT); free(BO); free(BC); free(SEMICOL); }

35. { <boolconst>.node-syn = TRUE.addr ; }

36. { <boolconst>.node-syn = FALSE.addr; }

37. { <id_num_rnum>.node-syn = ID.addr; }

38. { <id_num_rnum>.node-syn = NUM.addr; }

39. { <id_num_rnum>.node-syn = RNUM.addr; }

40. // REDUNDANT RULE

41. { <P1>.node_inh = ID.addr ;
<var_print>.node-Syn = <P1>.node-syn ; free(<P1>); }

42. { <var_print>.node-syn = NUM.addr ; }

43. { <var_print>.node-syn = RNUM.addr ; }

44. { <var_print>.node-syn = <boolconst>.node-syn ; free(<boolconst>); }

45. { <P1>.node-syn = make-node("ARRAY_", <P1>.node-inh ,
<new_index>.node-syn);

free(SQBO); free(<new_index>);
free(SQBC); }

46. { <P1>.node-syn = <P1>.node_inh ; free(eps); }

47. { <simplestmt>.node-syn = <assignstmt>.node-syn ; free(<assignstmt>); }

48. { <simplestmt>.node-Syn = <modReusestmt>.node-syn ; free(<modReusestmt>); }

49. { <whichstmt>.node_inh = ID.addr ;
<assignstmt>.node-syn = <whichstmt>.node-syn
free(<whichstmt>); }

50. { <lvalueIDstmt>.node_inh = <whichstmt>.node_inh ;
<whichstmt>.node_syn = <lvalueIDstmt>.node_syn ;
free (<lvalueIDstmt>); }

51. { <lvalueARRstmt>.node_inh = <whichStmt>.node_inh ;
<whichstmt>.node_syn = <lvalueARRstmt>.node_syn;
free (<lvalueARRstmt>); }

52. { <lvalueIDstmt>.node_syn = make_node ("ASSIGN", <lvalueIDStmt>.node_inh,
<exp>.node_syn);
free (<exp>);
free (ASSIGNOP);
free (SEMICOL); }

53. { <lvalueARRstmt>.node_syn = make_node ("ASSIGN", <element_index_with_expr>.node_syn
, <exp>.node_syn);
free (<exp>);
free (SQBO); free (SQBC); free (ASSIGNOP); free (SEMICOL);
free (<element_index_with_expr>); free (<exp>); }

54.

55. { <new_index>.node_syn = NUM.addr ; }

56. { <new_index>.node_syn = ID.addr ; }.

57. { <sign>.node_syn = PLUS.addr ; }

58. { <sign>.node_syn = MINUS.addr ; }

59. { <sign>.node_syn = NULL; }

60. { <modReusestmt>.node_syn = makenode ("MODULEREUSE", ID.addr,
<optional>.list_syn, <paralist>.list_syn);
free (<optional>);
free (USE); free (MODULE); free (WITH);
free (PARAMETERS); free (<paralist>); free (SEMICOL); }

61. { <paralist>.node-syn = make-node ("PARAMETER", <sign>.node-syn ,
<actual-para-list>. node-syn) ;
<paralist>. list-syn = insert-at-head (<paralist2>. list-syn , <paralist>. node-syn) ;
free (<sign>) ; free (<actual-para-list>); free (<paralist2>); }

62. { <paralist2>. node-syn = make-node ("PARAMETER", <sign>. node-syn ,
<actual-para-list>. node-syn);
<paralist2>. list-syn = insert-at-head (<paralist2>. list-syn, <paralist2>. node-syn);
free (COMMA) ; free (<sign>) ; free (<actual-para-list>); free (<paralist2>);

63. { <paralist2>. list-syn = NULL ; free (eps) ; }

64. { <actual-para-list>. node-syn = NUM. addr ; }

65. { <actual-para-list>. node-syn = RNUM. addr ; }

66. { <actual-para-list>. node-syn = <boolconst>. node-syn ; free (<boolconst>); }

67. { <actual-para-list>. node-syn = makenode ("PARAM_", ID. addr,
<N11>. node-syn) ;
free (<N11>) ; }

68. {<optional>. list-syn = <idList>. list-syn ; free (SQBO); free (SQBC) ;
free (ASSIGNOP); free (<idList>) ; } .

69. { <optional>. list-syn = NULL ;
free (eps) ; }

70. { <idList>. node-syn = ID. addr ;
<idList>. list-syn = insert-at-head (<N3>. list-syn, <idList>. node-syn)
free (<N3>); }

71. { <N3>. node-syn = ID. addr ;
<N3>. list-syn = insert-at-head (<N3>$_1$. list-syn , <N3>. node-syn);
free (<N3>$_1$); free (COMMA) ; }

72. { <N3>. list-syn = NULL ;
free (eps); }

73. { <exp>. node-syn = <arithOrboolExp>. node-syn ;
free (<arithOrboolExp>); }

74. { <exp>. node-syn = <U>. node-syn ; free (<U>); }

75. { <new-NT>. node-inh = <unary-op>. node-syn ;
<U>. node-syn = <new-NT>. node-syn;
free (<unary-op>); free (<new-NT>); }

76. { <unary-op>. node-syn = PLUS. addr; }

77. { <unary-op>. node-syn = MINUS . addr; }

78. { <new-NT>. node.syn = makenode ("UNARYASSIGN", <new-NT>. node-inh,
<arithExp>. node-syn);
free (BO);
free (<arithExp>);
free (BC); }

79. { <new-NT>. node-syn = makenode ("UNARY ASSIGN", <new-NT>. node-inh,
<id-num-rnum >. node-syn);
free (<id-num-rnum>); }

80. { <N4>. node-inh = <AnyTerm>. node-syn ;
<arithOrboolExp>, node-syn = <N4>. node-syn;
free (<Any Term>); free (<N4>); }

81. { $<N4>_1$. node-inh = makenode (<logop>.node-syn <N4>. node-inh , <AnyTerm>. node-syn);
<N4>. node-syn = $<N4>_1$. node-syn
free ($<N4>_1$),
free (<logOp>);
free (<AnyTerm>); }

82. { <NU>. node-syn = <N4>. node-inh; }

83. { <N5>. node-inh = <arithExp>. node-syn ;
<AnyTerm>. node-syn = <N5>. node-syn ;
free (<arithExp>);
free (<N5>); }

84. { <AnyTerm>. node-syn = <boolconst>. node-syn ; free (<boolconst>); }

85. { <N5>. node-syn = makenode (<relop>. node-syn, <N5>. node-inh ,
<arithExp>. node-syn);
free (<relop>);
free (<arithExp>); }

86. { <N5>.node_syn = <N5>.node_inh ; free(eps); }

87. { <N6>.node_inh = <term>.node_syn ;
<arithExp>.node_syn = <N6>.node_syn ;
free(<term>);
free(<N6>); }

88. { $<N6>_1$.node_inh = make_node(<op1>.node_syn, <N6>.node_inh, <term>.node_syn);
<N6>.node_syn = $<N6>_1$.node_syn ;
free(<op1>); free($<N6>_1$); free(<term>); }

89. { <N6>.node_syn = <N6>.node_inh ;
free(eps); }

90. { <N7>.node_inh = <factor>.node_syn ;
<term>.node_syn = <N7>.node_syn ;
free(<factor>);
free(<N7>); }

91. { $<N7>_1$.node_inh = make_node(<op2>.node_syn, <N7>.node_inh,
<factor>.node_syn);
<N7>.node_syn = $<N7>_1$.node_syn ;
free(<op2>);
free(<factor>);
free($<N7>_1$); }

92. { <N7>.node_syn = <N7>.node_inh ; free(eps); }

93. { <factor>.node_syn = <arithOrboolExp>.node_syn ;
free(BO); free(BC);
free(<arithOrboolExp>); }.

94. { <factor>.node_syn = NUM.addr ; }

95. { <factor>.node_syn = RNUM.addr ; }

96. { <N_11>.node_inh = ID.addr ;
<factor>.node_syn = <N_11>.node_syn ;
free(<N_11>); }

97. { <N_11>.node_syn = make_node("ARRAY_ACC", <N11> node_inh,
<element_index_with_expr>.node_syn);
free(SQBC); free(<element_index_with_expr>); free(SQBO); }

98. { <N_11>.node-syn = <N_11>.node-inh ; }

99. // REDUNDANT RULE :

100. { <N_10>.node-inh = <op1>.node-syn ;
<element_index_with_expr>.node-syn = <N_10>.node-syn
free (<op1>) ; free (<N_10>);

101. { <element_index_with_expr>.node-syn = <arrExpr>.node-syn ;
free (<arrExpr>); }

102. { <N_10>.node-syn = make_node ("UNI_NUMBER", <NL10>.node-inh ,
<new_index>.node-syn);
free (<new_index>); }

103. { <N_10>.node-syn = make_node ("UNI_ARR_NUM", <N_10>.node-inh ,
<arrExpr>.node-syn);
free (<arrExpr>);
free (BO);
free (BC);

104. { <arr_N4>.node-inh = <arrTerm>.node-syn ;
<arrExpr>.node-syn = <arr_N4>.node-syn ;
free (<arrTerm>); free (<arr_N4>); }

105. { $<arr\_N4>_1$.node-inh = make_node (<op1>.node-syn , <arr_N4>.node-inh ,
<arrTerm>.node-syn);
<arr_N4>.node-syn = $<arr\_N4>_1$.node-syn .
free (<arrTerm>); free ($<arr\_N4>_1$); }

106. { <arr_N4>.node-syn = <arr_N4>.node-inh ; }

107. { <arr_N5>.node-inh = <arrFactor>.node-syn ;
<arrTerm>.node-syn = <arr_N5>.node-syn ;
free (<arrFactor>);
free (<arr_N5>); }

108. { $<arr\_N5>_1$.node-inh = make_node (<op2>.node-syn , <arr_N5>.node-inh ,
<arr_Factor>.node-syn);
<arr_N5>.node-syn = $<arr\_N5>_1$.node-syn
free ($<arr\_N5>_1$); free (<arr_Factor>); }

109. { <arr-N5>.node-syn = <arr-N5>.node-inh ; }.

110. { <arrFactor>.node-syn = ID.addr ; }

111. { <arrFactor>.node-syn = NUM.addr ; }

112. { <arrFactor>.node-syn = <boolconst>.node-syn ; free (<boolconst>); }

113. { <arrFactor>.node-syn = <arrExpr>.node-syn ; free (BO); free (BC);
free (<arrExpr>); }

114. { <op1>.node-syn = PLUS.addr ; }

115. { <op1>.node-syn = MINUS.addr ; }

116. { <op2>.node-syn = MUL.addr ; }

117. { <op2>.node-syn = DIV.addr ; }

118. { <logOp>.node-syn = AND.addr ; }

119. { <logOp>.node-syn = OR.addr ; }

120. { <relOp>.node-syn = LT.addr ; }

121. { <rlOp>.node-syn = LE.addr ; }

22. { <rlop>.node-syn = GT.addr ; }

23. { <rlop>.node-syn = GE.addr ; }

24. { <rlop> node-syn = EQ.addr ; }

25. { <relop> node-syn = NE.addr ; }

126. { <declarestmt>.node-syn = make_node ("DECLARE", <idList>.list-syn ,
<dataType>.node-syn) ;
free (DECLARE); free (COLON); free (SEMICOL);
free (<idList>); free (<dataType>); }.

127. { <condstmt> .node-syn = make_node ("SWITCH", ID.addr ,
<case stmts>.list-syn , <default>.node-syn);
free (SWITCH); free (BC); free (<case stmts>); free (END); free (BO);
free (START); free (DEFAULT); }

128. { <casestmts>.node-syn = make_node ("CASE", <value>.node-syn ,
<stmts>.list-syn);
<caxstmts>.list-syn = insert_at_head (<N9>.list-syn, <case stmts>.node-syn);
free (CASE); free (<value>); free (COLON); free (<stmts>);
free (BREAK); free (SEMICOL); free (<N9>);

129. { <N9>.node_syn = make_node ("CASE", <value>.node_syn, <stmts>.list_syn);
<N9>.list_syn = insert_at_head (<N9>1.list_syn, <caseStmts>.node_syn);
free (CASE); free (<value>); free (COLON); free (<stmts>); free (BREAK);
free (SEMICOL); free (<N9>1); }

130. { <N9>.list_syn = NULL; free (eps); }

131. { <value>.node_syn = NUM.addr; }

132. { <value>.node_syn = TRUE.addr; }

133. { <value>.node_syn = FALSE.addr; }

134. { <dfault>.node_syn = make_node ("DEFAULTCASE", <stmts>.list_syn);
free (DEFAULT); free (BREAK); free (<stmts>); free (COLON);
free (SEMICOL); }

135. { <dfault>.node_syn = NULL; free (eps); }

136. { <iterStmt>.node_syn = make_node ("FOR", ID.addr,
<range_for_loop>.node_syn, <stmts>.list_syn);
free (FOR); free (BO); free (IN); free (<range_for_loop>); free (BC);
free (START); free (END); free (<stmts>);

37. { <iterStmt>.node_syn = make_node ("WHILE", <arthOrboolExp>.node_syn,
<stmts>.list_syn);
free (WHILE); free (BO); free (<arthOrboolExp>); free (BC);
free (START); free (<stmts>); free (END); }

38. { <range_for_loop>.node_syn = make_node ("FOR RANGE",
<index_forLoop>1.node_syn,
<index_forloop>2.node_syn);
free (<index_forloop>1);
free (<index_forloop>2);
free (RANGEOP); }

139. { <index_forloop>.node_syn = make_node ("FOR INDEX",
<sign>.node_syn, <new_index_forloop>.node_syn);
free (<sign_forloop>); free (<new_index_forloop>); }

140. { <new_index_forloop>.node_syn = NUM.addr; }

141. { <sign_forloop>.node_syn = PLUS.addr; }

142. { <sign_forloop>.node_syn = MINUS.addr; }

143. {<sign_forloop>.node_syn = NULL; free(eps); }.