1. <finalProgram> -> <program> EOF
{
   // GOING DOWN
   <finalProgram>.addr = <program>.addr
   // GOING UP
   free(EOF)

}

2. <program> -> <moduleDeclarations> <otherModules> <driverModule> <otherModules>
{
   // GOING UP
   <program>.addr = createLinkedList(
<moduleDeclarations>.list_addr_syn,<otherModules>.list_addr_syn,<driverModule>.addr,<other
Modules>.list_addr_syn)

}

3. <moduleDeclarations> -> <moduleDeclaration> <moduleDeclarations1>
{
   //GOING UP
   <moduleDeclarations>.list_addr_syn =
insertAtBegin(<moduleDeclaration>.addr,<moduleDeclarations1>.list_addr_syn)
   <moduleDeclarations>.addr = <moduleDeclarations>.list_addr_syn
}

4. <moduleDeclarations> -> epsilon
{
   // GOING UP
   <moduleDeclarations>.list_addr_syn = NULL
   free(epsilon)
}

5. <moduleDeclaration> -> DECLARE MODULE ID SEMICOL
{
   // GOING UP
   <moduleDeclaration>.addr = ID.addr
   free(DECLARE)
   free(MODULE)
   free(SEMICOL)
}

6. <otherModules> -> <module> <otherModules1>
{

```
    // GOING UP
    <otherModules>.list_addr_syn =
insertAtBegin(<module>.addr,<otherModules1>.list_addr_syn)
    <otherModules>.addr = <otherModules>.list_addr_syn
}

7. <otherModules> -> epsilon
{
    // GOING UP
    <otherModules>.list_addr_syn = NULL
    free(epsilon)
}

8. <driverModule> -> DRIVERDEF DRIVER PROGRAM DRIVERENDDEF <moduleDef>
{
    // GOING DOWN
    <driverModule>.addr = <moduleDef>.addr
    // GOING UP
    free(DRIVERDEF)
    free(DRIVERENDDEF)
    free(PROGRAM)
    free(DRIVER)
}

9. <module> -> DEF MODULE ID ENDDEF TAKES INPUT SQBO <input_plist> SQBC
SEMICOL <ret> <moduleDef>
{
    // GOING DOWN
    <module>.addr = makeNode(ID,<module>.addr_syn,<moduleDef>.addr)
    // GOING UP
    free(DEF)
    free(MODULE)
    free(ID)
    free(ENDDEF)
    free(TAKES)
    free(INPUT)
    free(SQBO)
    free(SQBC)
    free(SEMICOL)
    <module>.addr_syn = makeNode(PARAMETERS,<input_plist>.addr,<ret>.addr)
}

10. <ret> -> RETURNS SQBO <output_plist> SQBC SEMICOL
{
```

```
    // GOING DOWN
    <ret>.addr = <output_plist>.addr
    // GOING UP
    free(RETURNS)
    free(SQBO)
    free(SQBC)
    free(SEMICOL)
}

11. <ret> -> epsilon
{
    // GOING DOWN
    free(epsilon)
}

12. <input_plist> -> ID COLON <dataType> <leftFactored_input_plist>
{
    // GOING DOWN
    <input_plist>.addr = makeNode(ID,<dataType>.addr,NULL)
    // GOING UP
    <input_plist>.list_addr_syn =
insertAtBegin(<input_plist>.addr,<leftFactored_input_plist>.list_addr_syn)
    free(COLON)
    free(ID)
}

13. <leftFactored_input_plist> -> COMMA ID COLON <dataType> <leftFactored_input_plist1>
{
    // GOING DOWN
    <leftFactored_input_plist>.addr = makeNode(ID,<dataType>.addr,NULL)
    // GOING UP
    <leftFactored_input_plist>.list_addr_syn =
insertAtBegin(<leftFactored_input_plist>.addr,<leftFactored_input_plist1>.list_addr_syn)
    free(COLON)
    free(COMMA)
    free(ID)

}

14. <leftFactored_input_plist> -> epsilon
{
    // GOING DOWN
    <leftFactored_input_plist>.list_addr_syn = NULL
    // GOING UP
```

```
    free(epsilon)
}

15. <output_plist> -> ID COLON <type> <leftFactored_output_plist>
{
    // GOING DOWN
    <output_plist>.addr = makeNode(ID, <type>.addr, NULL)
    // GOING UP
    <output_plist>.list_addr_syn =
insertAtBegin(<output_plist>.addr,<leftFactored_output_plist>.list_addr_syn)
    free(COLON)
    free(ID)
}

16. <leftFactored_output_plist> -> COMMA ID COLON <type> <leftFactored_output_plist>
{
    // GOING DOWN
    <leftFactored_output_plist>.addr = makeNode(ID, <type>.addr, NULL)
    // GOING UP
    free(COMMA)
    free(COLON)
    free(ID)
}

17. <leftFactored_output_plist> -> epsilon
{
    // GOING DOWN
    <leftFactored_output_plist>.addr = NULL
    // GOING UP
    free(epsilon)
}

18. <dataType> -> INTEGER
{
    // GOING DOWN
    <dataType>.addr = INTEGER.addr
}

19. <dataType> -> REAL
{
    // GOING DOWN
    <dataType>.addr = REAL.addr
}
```

20. <dataType> -> BOOLEAN
{
   // GOING DOWN
   <dataType>.addr = BOOLEAN.addr
}

21. <dataType> -> ARRAY SQBO <arrRange> SQBC OF <type>
{
   // GOING DOWN
   <dataType>.addr = makeNode(ARRAY,<type>.addr,<arrRange>.addr_syn)
   // GOING UP
   free(ARRAY)
   free(SQBO)
   free(SQBC)
   free(OF)
}

22. <arrRange> -> <sign> <leftFactored_arrRange>
{
   // GOING DOWN
   <arrRange>.addr = <sign>.addr
   <leftFactored_arrRange>.addr_inh = <arrRange>.addr
   // GOING UP
   <arrRange>.addr_syn = <leftFactored_arrRange>.addr
}

23. <leftFactored_arrRange> -> <idNum1> RANGEOP <sign> <idNum2>
{
   // GOING DOWN
   <leftFactored_arrRange>.addr = makeNode(RANGEOP,<idNum1>.addr,<idNum2>.addr)
   <idNum1>.addr_inh = <leftFactored_arrRange>.addr_inh
   <idNum2>.addr_inh = <sign>.addr
   // GOING UP
   free(RANGEOP)
}

24. <idNum> -> ID
{
   // GOING DOWN
   <idNum>.addr = makeNode(ID,<idNum>.addr_inh,NULL)
   // GOING UP
   free(ID)
}

25. <idNum> -> NUM
{
   // GOING DOWN
   <idNum>.addr = makeNode(NUM,<idNum>.addr_inh,NULL)
   // GOING UP
   free(NUM)
}

26. <sign> -> <pm>
{
   // GOING DOWN
   <sign>.addr = <pm>.addr
}

27. <sign> -> epsilon
{
   // GOING DOWN
   <sign>.addr = NULL
   // GOING UP
   free(epsilon)
}

28. <type> -> INTEGER
{
   // GOING DOWN
   <type>.addr = INTEGER.addr
}

29. <type> -> REAL
{
   // GOING DOWN
   <type>.addr = REAL.addr
}

30. <type> -> BOOLEAN
{
   //GOING DOWN
   <type>.addr = BOOLEAN.addr
}

31. <moduleDef> -> START <statements> END
{
   // GOING DOWN
   <moduleDef>.addr = <statements>.addr

```
}

32. <statements> -> <statement> <statements1>
{
   // GOING UP
   <statements>.list_addr_syn = insertAtBegin(<statement>.addr,<statements1>.list_addr_syn)
   <statements>.addr = <statements>.list_addr_syn
}

33. <statements> -> epsilon
{
   // GOING DOWN
   <statements>.list_addr_syn = NULL
}

34. <statement> -> <ioStmt>
{
   // GOING DOWN
   <statement>.addr = <ioStmt>.addr
}

35. <statement> -> <simpleStmt>
{
   // GOING DOWN
   <statement>.addr -> <simpleStmt>.addr
}

36. <statement> -> <declareStmt>
{
   // GOING DOWN
   <statement>.addr = <declareStmt>.addr
}

37. <statement> -> <conditionalStmt>
{
   // GOING DOWN
   <statement>.addr = <conditionalStmt>.addr
}

38. <statement> -> <iterativeStmt>
{
   // GOING DOWN
   <statement>.addr = <iterativeStmt>.addr
}
```

39. <ioStmt> -> GET_VALUE BO ID BC SEMICOL
{
    // GOING DOWN
    <ioStmt>.addr = makeNode(GET_VALUE,ID.addr,NULL)
    // GOING UP
    free(GET_VALUE)
    free(BO)
    free(BC)
    free(SEMICOL)
}

40. <ioStmt> -> PRINT BO <leftFactored_ioStmt>
{
    // GOING DOWN
    <ioStmt>.addr = makeNode(PRINT,<leftFactored_ioStmt>.addr)
    // GOING UP
    free(BO)
    free(PRINT)
}

41. <leftFactored_ioStmt> -> <var> BC SEMICOL
{
    // GOING DOWN
    <leftFactored_ioStmt>.addr = <var>.addr
    // GOING UP
    free(BC)
    free(SEMICOL)
}

42. <leftFactored_ioStmt> -> <boolValues> BC SEMICOL
{
    // GOING DOWN
    <leftFactored_ioStmt>.addr = <boolValues>.addr
    // GOING UP
    free(BC)
    free(SEMICOL)
}

43. <boolValues> -> true
{
    // GOING DOWN
    <boolValues>.addr = true.addr
}

44. <boolValues> -> false
{
    // GOING DOWN
    <boolValues>.addr = false.addr
}

45. <var> -> ID <whichId>
{
    // GOING DOWN
    <var>.addr = makeNode(ID, <whichId>.addr_syn, <whichId>.addr)
    // GOING UP
    free(ID)
}

46. <var> -> NUM
{
    // GOING DOWN
    <var>.addr = NUM.addr
}

47. <var> -> RNUM
{
    // GOING DOWN
    <var>.addr = RNUM.addr
}

48. <whichId> -> SQBO <sign> <leftFactored_whichId>
{
    // GOING DOWN
    <whichId>.addr = <leftFactored_whichId>.addr
    // GOING UP
    <whichId>.addr_syn = <sign>.addr
    free(SQBO)
}

49. <whichId> -> epsilon
{
    // GOING DOWN
    <whichId>.addr = NULL
    // GOING UP
    free(epsilon)
}

50. <leftFactored_whichId> -> ID SQBC
{
   // GOING DOWN
   <leftFactored_whichId>.addr = ID.addr
   // GOING UP
   free(SQBC)
}

51. <leftFactored_whichId> -> NUM SQBC
{
   // GOING DOWN
   <leftFactored_whichId>.addr = NUM.addr
   // GOING UP
   free(SQBC)
}

52. <simpleStmt> -> <assignmentStmt>
{
   // GOING DOWN
   <simpleStmt>.addr = <assignmentStmt>.addr
}

53. <simpleStmt> -> <moduleReuseStmt>
{
   // GOING DOWN
   <simpleStmt>.addr = <moduleReuseStmt>.addr
}

54. <assignmentStmt> -> ID <whichStmt>
{
   // GOING DOWN
   <assignmentStmt>.addr =
makeNode(ASSIGNOP,<assignmentStmt>.addr_syn,<whichStmt>.addr)
   <whichStmt>.addr_inh = ID.addr
   // GOING UP
   <assignmentStmt>.addr_syn = <whichId>.addr_syn
}

55. <whichStmt> -> <lvalueIDStmt>
{
   // GOING DOWN
   <whichStmt>.addr = <lvalueIDStmt>.addr
   // GOING UP
   <whichStmt>.addr_syn = <whichStmt>.addr_inh

}

56. <whichStmt> -> <lvalueARRStmt>
{
   // GOING DOWN
   <whichStmt>.addr = <lvalueARRStmt>.addr
   <lvalueARRStmt>.addr_inh = <whichStmt>.addr_inh
   // GOING UP
   <whichStmt>.addr_syn = <lvalueARRStmt>.addr_syn
}

57. <lvalueIDStmt> -> ASSIGNOP <expression> SEMICOL
{
   // GOING DOWN
   <lvalueIDStmt>.addr = <expression>.addr
   // GOING UP
   free(ASSIGNOP)
   free(SEMICOL)
}

58. <lvalueARRStmt> -> SQBO <arithmeticExprWArr> SQBC ASSIGNOP <expression>
SEMICOL
{
   // GOING DOWN
   <lvalueARRStmt>.addr = <expression>.addr
   <lvalueARRStmt>.addr_syn =
makeNode(<lvalueARRStmt>.addr_inh,<arithmeticExprWArr>.addr,NULL)
   // GOING UP
   free(SQBO)
   free(SQBC)
   free(ASSIGNOP)
   free(SEMICOL)
}

59. <moduleReuseStmt> -> <optional> USE MODULE ID WITH PARAMETERS <paramList>
SEMICOL
{
   // GOING DOWN
   <moduleReuseStmt>.addr = makeNode(ID,<optional>.addr,<paramList>.addr)
   // GOING UP
   free(USE)
   free(MODULE)
   free(WITH)
   free(PARAMETERS)

```
    free(SEMICOL)
}

60. <param> -> <sign> <signedParam>
{
    // GOING DOWN
    <param>.addr = makeNode(<signedParam>.addr,<sign>.addr,<signedParam>.addr_syn)
}

61. <signedParam> -> ID <arrID>
{
    // GOING DOWN
    <signedParam>.addr = ID.addr
    <signedParam>.addr_syn = <arrID>.addr
}

62. <arrID> -> SQBO <arithmeticExprWArr> SQBC
{
    // GOING DOWN
    <arrID>.addr = <arithmeticExprWArr>.addr
    // GOING UP
    free(SQBO)
    free(SQBC)
}

63. <arrID> -> epsilon
{
    // GOING DOWN
    <arrID>.addr = NULL
    // GOING UP
    free(epsilon)
}

64. <signedParam> -> NUM
{
    // GOING DOWN
    <signedParam>.addr = NUM.addr
}

65. <signedParam> -> RNUM
{
    // GOING DOWN
    <signedParam>.addr = RNUM.addr
}
```

66. <param> -> <boolValues>
{
   // GOING DOWN
   <param>.addr = <boolValues>.addr
}

67. <paramList> -> <param> <leftFactored_paramList>
{
   // GOING UP
   <paramList>.list_addr_syn = insertAtBegin(<param>.addr,
<leftFactored_paramList>.list_addr_syn)
   <paramList>.addr = <paramList>.list_addr_syn
}

68. <leftFactored_paramList> -> COMMA <param> <leftFactored_paramList1>
{
   // GOING UP
   <leftFactored_paramList>.list_addr_syn = insertAtBegin(<param>.addr,
<leftFactored_paramList1>.list_addr_syn)
   free(COMMA)
}

69. <leftFactored_paramList> -> epsilon
{
   // GOING DOWN
   <leftFactored_paramList>.list_addr_syn = NULL
   // GOING UP
   free(epsilon)
}

70. <optional> -> SQBO <idList> SQBC ASSIGNOP
{
   // GOING DOWN
   <optional>.addr = <idList>.addr
   // GOING UP
   free(SQBO)
   free(SQBC)
   free(ASSIGNOP)
}

71. <optional> -> epsilon
{
   // GOING DOWN

```
    <optional>.addr = NULL
    // GOING UP
    free(epsilon)
}

72. <idList> -> ID <leftFactored_idList>
{
    // GOING DOWN
    <idList>.addr = ID.addr
    // GOING UP
    <idList>.list_addr_syn = insertAtBegin(ID, <leftFactored_idList>.list_addr_syn)
}

73. <leftFactored_idList> -> COMMA ID <leftFactored_idList>
{
    // GOING UP
    <leftFactored_idList>.list_addr_syn = insertAtBegin(ID, <leftFactored_idList>.list_addr_syn)
    free(COMMA)
}

74. <leftFactored_idList> -> epsilon
{
    // GOING DOWN
    <leftFactored_idList>.list_addr_syn = NULL
    // GOING UP
    free(epsilon)
}

75. <expression> -> <arithmeticBooleanExpr>
{
    // GOING DOWN
    <expression>.addr = <arithmeticBooleanExpr>.addr_syn
}

76. <expression> -> <unaryTerm>
{
    // GOING DOWN
    <expression>.addr = <unaryTerm>.addr
}

77. <unaryTerm> -> <pm> <arithmeticFactor>
{
    // GOING DOWN
    <unaryTerm>.addr = makeNode(<pm>.addr,<arithmeticFactor>.addr,NULL)
```

}

78. <arithmeticFactor> -> BO <arithmeticExpr> BC
{
   arithmeticFactor.addr = arithmeticExpr.addr_syn ;
   free(BO)
   free(BC)
}

79. <arithmeticFactor> -> ID
{
   // GOING DOWN
   <arithmeticFactor>.addr = ID.addr
}

80. <arithmeticFactor> -> NUM
{
   // GOING DOWN
   <arithmeticFactor>.addr = NUM.addr
}

81. <arithmeticFactor> -> RNUM
{
   // GOING DOWN
   <arithmeticFactor>.addr = RNUM.addr
}

82. <arithmeticBooleanExpr> -> <anyTerm> <logicalTerm>
{
   // GOING DOWN
   <arithmeticBooleanExpr>.addr = <anyTerm>.addr
   <logicalTerm>.addr_inh = <arithmeticBooleanExpr>.addr
   // GOING UP
   <arithmeticBooleanExpr>.addr_syn = <logicalTerm>.addr_syn
}

83. <logicalTerm> -> <logicalOp> <anyTerm> <logicalTerm1>
{
   // GOING DOWN
   <logicalTerm>.addr = makeNode(<logicalOp>.addr,<logicalTerm>.addr_inh,<anyTerm>.addr)
   <logicalTerm1>.addr_inh = logicalTerm.addr
   // GOING UP
   <logicalTerm>.addr_syn = <logicalTerm1>.addr_syn
}

84. <logicalTerm> -> epsilon
{
   // GOING DOWN
   <logicalTerm>.addr_syn = <logicalTerm>.addr_inh
   free(epsilon)
}

85. <anyTerm> -> <arithmeticExpr> <relationalTerm>
{
   // GOING DOWN
   <anyTerm>.addr_syn = <arithmeticExpr>.addr_syn
   <relationalTerm>.addr_inh = <anyTerm>.addr_syn
   // GOING UP
   <anyTerm>.addr = <relationalTerm>.addr
}

86. <relationalTerm> -> <relationalOp> <arithmeticExpr>
{
   // GOING DOWN
   <relationalTerm>.addr =
makeNode(<relationalOp>.addr,<relationalTerm>.addr_inh,<arithmeticExpr>.addr_syn)
}

87. <relationalTerm> -> epsilon
{
   // GOING DOWN
   <relationalTerm>.addr = NULL
   // GOING UP
   free(epsilon)
}

88. <arithmeticExpr> -> <term> <leftFactored_arithmeticExpr>
{
   // GOING DOWN
   <arithmeticExpr>.addr = <term>.addr ;
   <leftFactored_arithmeticExpr>.addr_inh = <arithmeticExpr>.addr ;
   // GOING UP
   <arithmeticExpr>.addr_syn = <leftFactored_arithmeticExpr>.addr_syn ;
}

89. <leftFactored_arithmeticExpr> -> <pm> <term> <leftFactored_arithmeticExpr1>
{
   // GOING DOWN

```
   <leftFactored_arithmeticExpr>.addr = makeNode(<pm>.addr ,
<leftFactored_arithmeticExpr>.addr_inh , <term>.addr_syn)
   <leftFactored_arithmeticExpr1>.addr_inh = <leftFactored_arithmeticExpr>.addr
   // GOING UP
   <leftFactored_arithmeticExpr>.addr_syn <leftFactored_arithmeticExpr1>.addr_syn

}

90. <leftFactored_arithmeticExpr> -> epsilon
{
   // GOING DOWN
   <leftFactored_arithmeticExpr>.addr_syn = <leftFactored_arithmeticExpr>.addr_inh ;
   // GOING UP
   free(epsilon)
}

91. <pm> -> PLUS
{
   // GOING DOWN
   <pm>.addr = PLUS.addr;
}

92. <pm> -> MINUS
{
   // GOING DOWN
   <pm>.addr = MINUS.addr;
}

93. <md> -> MUL
{
   // GOING DOWN
   <md>.addr = MUL.addr ;
}

94. <md> -> DIV
{
   // GOING DOWN
   <md>.addr = DIV.addr ;
}

95. <term> -> <factor> <leftFactored_term>
{
   // GOING DOWN
   <term>.addr = <factor>.addr ;
```

```
    <leftFactored_term>.addr_inh = <term>.addr ;
    // GOING UP
    <term>.addr_syn = <leftFactored_term>.addr_syn;
}

96. <leftFactored_term> -> <md> <factor> <leftFactored_term1>
{
    // GOING DOWN
    <leftFactored_term>.addr = makeNode(<md>.addr, <leftFactored_term>.addr_inh ,
<factor>.addr);
    <leftFactored_term1>.addr_inh = <leftFactored_term>.addr ;

    // GOING UP
    <leftFactored_term>.addr_syn = <leftFactored_term1>.addr_syn
}

97. <leftFactored_term> -> epsilon
{
    // GOING DOWN
    <leftFactored_term>.addr_syn = <leftFactored_term>.addr_inh
    // GOING UP
    free(epsilon)
}

98. <factor> -> BO <arithmeticBooleanExpr> BC
{
    // GOING DOWN
    <factor>.addr = <arithmeticBooleanExpr>.addr
    // GOING UP
    free(BO)
    free(BC)
}

99. <factor> -> <param>
{
    // GOING DOWN
    <factor>.addr = <param>.addr
}

100. <arithmeticExprWArr> -> <termWArr> <leftFactored_arithmeticExprWArr>
{
    // GOING DOWN
    <arithmeticExprWArr>.addr = <termWArr>.addr ;
    <leftFactored_arithmeticExprWArr>.addr_inh = <arithmeticExprWArr>.addr ;
```

```
    // GOING UP
    <arithmeticExprWArr>.addr_syn = <leftFactored_arithmeticExprWArr>.addr_syn ;
}

101. <leftFactored_arithmeticExprWArr> -> <pm> <termWArr>
<leftFactored_arithmeticExprWArr1>
{
    // GOING DOWN
    <leftFactored_arithmeticExprWArr>.addr = makeNode(<pm>.addr ,
<leftFactored_arithmeticExprWArr>.addr_inh , <termWArr>.addr_syn)
    <leftFactored_arithmeticExprWArr1>.addr_inh = <leftFactored_arithmeticExpr>.addr
    // GOING UP
    <leftFactored_arithmeticExprWArr>.addr_syn <leftFactored_arithmeticExprWArr1>.addr_syn

}

102. <leftFactored_arithmeticExprWArr> -> epsilon
{
    // GOING DOWN
    <leftFactored_arithmeticExprWArr>.addr_syn = <leftFactored_arithmeticExprWArr>.addr_inh
;
    // GOING UP
    free(epsilon)
}

103. <termWArr> -> <factorWArr> <leftFactored_termWArr>
{
    // GOING DOWN
    <termWArr>.addr = <signedFactorWArr>.addr ;
    <leftFactored_termWArr>.addr_inh = <termWArr>.addr ;
    // GOING UP
    <termWArr>.addr_syn = <leftFactored_termWArr>.addr_syn;
}

104. <leftFactored_termWArr> -> <md> <signedFactorWArr> <leftFactored_termWArr1>
{
    // GOING DOWN
    <leftFactored_termWArr>.addr = makeNode(<md>.addr, <leftFactored_termWArr>.addr_inh ,
<signedFactorWArr>.addr);
    <leftFactored_termWArr1>.addr_inh = <leftFactored_termWArr>.addr ;

    // GOING UP
    <leftFactored_termWArr>.addr_syn = <leftFactored_termWArr1>.addr_syn
```

```
}
105. <leftFactored_termWArr> -> epsilon
{
   // GOING DOWN
   <leftFactored_termWArr>.addr_syn = <leftFactored_termWArr>.addr_inh
   // GOING UP
   free(epsilon)

}
106. <signedFactorWArr> -> <sign> <factorWArr>
{
   // GOING DOWN
   <signedFactorWArr>.addr = makeNode(<signedFactorWArr>, <sign>.addr,
<factorWArr>.addr)
}

107. <factorWArr> -> BO <arithmeticExprWArr> BC
{
   // GOING DOWN
   <factorWArr>.addr = <arithmeticExprWArr>.addr_syn
   // GOING UP
   free(BO)
   free(BC)
}

108. <factorWArr> -> ID
{
   // GOING DOWN
   <factorWArr>.addr= ID.addr
}

109. <factorWArr> -> RNUM
{
   // GOING DOWN
   <factorWArr>.addr= RNUM.addr
}

110. <factorWArr> -> NUM
{
   // GOING DOWN
   <factorWArr>.addr= NUM.addr
}

111. <logicalOp> -> AND
```

```
{
   // GOING DOWN
   <logicalOp>.addr = AND.addr
}

112. <logicalOp> -> OR
{
   // GOING DOWN
   <logicalOp>.addr = OR.addr
}

113. <relationalOp> -> LT
{
   // GOING DOWN
   <relationalOp>.addr = LT.addr
}

114. <relationalOp> -> LE
{
   // GOING DOWN
   <relationalOp>.addr =LE.addr
}

115. <relationalOp> -> GT
{
   // GOING DOWN
   <relationalOp>.addr = GT.addr
}

116. <relationalOp> -> GE
{
   // GOING DOWN
   <relationalOp>.addr = GE.addr
}

117. <relationalOp> -> EQ
{
   // GOING DOWN
   <relationalOp>.addr = EQ.addr
}

118. <relationalOp> -> NE
{
   // GOING DOWN
```

```
    <relationalOp>.addr = NE.addr
}

119. <declareStmt> -> DECLARE <idList> COLON <dataType> SEMICOL
{
    // GOING DOWN
    <declareStmt>.addr = makeNode(DECLARE,<idList>.addr,<dataType>.addr)

}

120. <conditionalStmt> -> SWITCH BO ID BC START <caseStmt> <dfault> END
{
    // GOING DOWN
    <conditionalStmt>.addr = <caseStmt>.addr
    // GOING UP
    <conditionalStmt>.list_addr_syn = <caseStmt>.list_addr_syn
    <dfault>.list_addr_inh = <conditionalStmt>.list_addr_syn
}

121. <caseStmt> -> CASE <value> COLON <statements> BREAK SEMICOL
<leftFactored_caseStmt>
{
    // GOING DOWN
    <caseStmt>.addr = makeNode(<value>.addr , <statements>.addr)

    // GOING UP
    <caseStmt>.list_addr_syn = insertAtBegin(<caseStmt>.addr ,
<leftFactored_caseStmt>.list_addr_syn)
    free(CASE)
    free(COLON)
    free(BREAK)
    free(SEMICOL)
}

122. <leftFactored_caseStmt> -> CASE <value> COLON <statements> BREAK SEMICOL
<leftFactored_caseStmt1>
{
    // GOING DOWN
    <leftFactored_caseStmt>.addr = makeNode(<value>.addr , <statements>.addr)
    free(CASE)
    free(COLON)
    free(BREAK)
    free(SEMICOL)
```

```
    # doubt here
    // GOING UP
    <leftFactored_caseStmt>.list_addr_syn = insertAtBegin(<caseStmt>.addr ,
<leftFactored_caseStmt1>.list_addr_syn)

}

123. <leftFactored_caseStmt> -> epsilon
{
    // GOING DOWN
    <leftFactored_caseStmt>.list_addr_syn=NULL
    free(epsilon)
}

124. <value> -> NUM
{
    // GOING DOWN
    <value>.addr = NUM.addr
}

125. <value> -> <boolValues>
{
    // GOING DOWN
    <value>.addr = <boolValues>.addr
}

126. <dfault> -> DEFAULT COLON <statements> BREAK SEMICOL
{
    // GOING DOWN
    <dfault>.addr = makeNode(DEFAULT,<statements>.addr)
    insertAtEnd(<dfault>.addr,<dfault>.list_addr_inh)
    free(DEFAULT)
    free(COLON)
    free(BREAK)
    free(SEMICOL)
}

127. <dfault> -> epsilon
{
    // GOING DOWN
    insertAtEnd(NULL,<dfault>.list_addr_inh)
    free(epsilon)
}
```

128. <iterativeStmt> -> FOR BO ID IN <sign1> NUM1 RANGEOP <sign2> NUM2 BC START <statements> END
{
   // GOING DOWN
   <iterativeStmt>.addr = makeNode(FOR,makeNode(ID,makeNode(NUM1,<sign1>.addr,NULL),makeNode(NUM2,<sign2>.addr,NULL)),<statements>.addr)
}

129. <iterativeStmt> -> WHILE BO <arithmeticBooleanExpr> BC START <statements> END
{
   // GOING DOWN
   <iterativeStmt>.addr = makeNode(WHILE,<arithmeticBooleanExpr>.addr,<statements>.addr)
}