1.
<program> -> <moduleDeclarations> <otherModules1> <driverModule> TK_END <otherModules2>[
    1. <program>.node_syn = make_node("PROGRAM", make_node("MODULEDECLARATIONS",MODULEDECLARATIONS.list_head_syn),
                                        make_node("MODULEDEFINITIONS", <otherModules1>.list_head_syn),
                                        DRIVERMODULE.node_syn,
                                        make_node("MODULEDEFINITIONS", <otherModules2>.list_head_syn)
)
    2. free(<moduleDeclarations>)
    3. free(OTHERMODULES1)
    4. free(TK_END)
    5. free(DRIVERMODULE)
    6. free(OTHERMODULES2)
]
2. <moduleDeclarations> -> <moduleDeclaration> TK_SEMICOL <moduleDeclarations1>
{
    1. MODULEDECLARATIONS.list_head_syn = insert_at_head(MODULEDECLARATIONS1.list_head_syn, MODULEDECLARATION.node_syn)
    2
    . free(MODULEDECLARATION)
    3. free(TK_SEMICOL)
    3. free(MODULEDECLARATIONS1)
}

3. <moduleDeclarations> -> epsilon
{
    1. MODULEDECLARATIONS.list_head_syn = NULL
}

4. <moduleDeclaration> -> TK_DECLARE TK_MODULE TK_ID
{
    1. MODULEDECLARATION.node_syn = id.addr
    2. free(declare)
    3. free(module)
}

5. <otherModules> -> <module> TK_END <otherModules1> // agar othermodules1.list_head_syn null hua to? same doubt rule 13//null insert hoga to

{
    1. OTHERMODULES.list_head_syn = insert_at_head(OTHERMODULES1.list_head_syn, MODULE.node_syn)
    2. free(MODULE)
    3. free(TK_END)
    4. free(OTHERMODULES1)
}

6.<otherModules> -> epsilon
{

1. OTHERMODULES.list_head_syn = NULL
}

7. <driverModule> -> TK_DRIVERDEF TK_DRIVER TK_PROGRAM TK_DRIVERENDDEF <moduleDef>
// singh coder ne node bnaya hai// node banega
{
1. DRIVERMODULE.node_syn = make_node("DRIVER",moduleDef.node_syn)
2. free(driverdef)
3. free(driver)
4. free(program)
5. free(driverenddef)
6. free(MODULEDEF)
}

8. <module> -> TK_DEF TK_MODULE TK_ID TK_ENDDEF TK_TAKES TK_INPUT TK_SQBO <input_pli
st> TK_SQBC TK_SEMICOL <ret> <moduleDef>
{
1. MODULE.node_syn = make_node("MODULE",
id.addr,
make_node("INPUT_PARAMETERS_LIST", INPUT_PLIST.list_head_syn),
RET.node_syn,
MODULEDEF.node_syn)
2. free(def)
3. free(module)
4. free(enddef)
5. free(takes)
6. free(input)
7. free(sqbo)
8. free(INPUT_PLIST)
9. free(sqbc)
10. free(semicol)
11. free(RET)
12. free(MODULEDEF)
}

9. <ret> -> TK_RETURNS TK_SQBO <output_plist> TK_SQBC TK_SEMICOL
{
1. RET.node_syn = make_node("OUTPUT_PARAMETERS_LIST", OUTPUT_PLIST.list_head_syn)
2. free(returns)
3. free(sqbo)
4. free(OUTPUT_PLIST)
5. free(sqbc)
6. free(semicol)
}

10. <ret> -> epsilon
{
1. RET.node_syn = make_node("OUTPUT_PARAMETERS_LIST", NULL)

}

11. <input_plist> -> TK_ID TK_COLON <dataType> <N1>
{
Input_plist.node_syn=make_node("inp_parameter",Id.addr,datatype.node_syn)
input_plist.list_head_syn = insert_at_head(N1.list_head_syn,input_plist.node_syn)
free(colon)
free(dataType)
free(N1)

}

12. <N1> -> TK_COMMA TK_ID TK_COLON <dataType> <N1'>
{
N1.node_syn=make_node("inp_parameter",Id.addr,datatype.node_syn)
N1.list_head_syn = insert_at_head(N1'.list_head_syn,N1.node_syn)
free(comma)
free(colon)
free(DATATYPE)
free(N1')
}

13. <N1> -> epsilon
{
1. N1.list_head_syn = NULL
}

14. <output_plist> -> TK_ID TK_COLON <type> <N2>
{
output_plist.node_syn=make_node("out_parameter",ID.addr,type.node_syn)
output_plist.list_head_syn = insert_at_head(N2.list_head_syn,output_plist.node_syn)
free(colon)
free(Type)
free(N2)
}

15. <N2> -> TK_COMMA TK_ID TK_COLON <type> <N2'>
{
N2.node_syn=make_node("out_parameter",ID.addr,type.node_syn)
N2.list_head_syn = insert_at_head(N2'.list_head_syn,N2.node_syn)
free(comma)
free(colon)
free(TYPE)
free(N2')
}

16 <N2> -> epsilon
{
N2.list_head_syn = NULL

}

17. <dataType> -> TK_INTEGER
{
   1. DATATYPE.node_syn = integer.addr
}

18. <dataType> -> TK_REAL
{
   1. DATATYPE.node_syn = real.addr
}

19. <dataType> -> TK_BOOLEAN
{
   1. DATATYPE.node_syn = boolean.addr
}

20. <dataType> -> TK_ARRAY TK_SQBO <range_arrays> TK_SQBC TK_OF <type>
{
   1. DATATYPE.node_syn = make_node("ARRAY", RANGE_ARRAYS.node_syn, TYPE.node_syn)
   2. free(array)
   3. free(sqbo)
   4. free(RANGE_ARRAYS)
   5. free(sqbc)
   6. free(of)
   7. free(TYPE)
}
21. <range_arrays> -> <index_arr1> TK_RANGEOP <index_arr2> {
    RANGE_ARRAYS.node_syn = make_node("range_arrays",index_arr1.node_syn,index_arr2.node_sy
n)
    free(index_arr1)
    free(RANGEOP)
    free(index_arr2)
}

22. <type> -> TK_INTEGER
{
   1. TYPE.node_syn = integer.addr
}

23.<type> -> TK_REAL
{
   1. TYPE.node_syn = real.addr
}

24. <type> -> TK_BOOLEAN
{
   1. TYPE.node_syn = boolean.addr
}

25. <moduleDef> -> TK_START <statements>
{
   1. MODULEDEF.node_syn = make_node("STATEMENTS", STATEMENTS.list_head_syn)
   2. free(start)
   3. free(STATEMENTS)

}
26. <statements> -> <iterativeStmt> TK_END <statements1>
{
   1. STATEMENTS.list_head_syn = insert_at_head(STATEMENTS1.list_head_syn,ITERATIVESTATEMENT.node_syn);
   free(iterativeStmt)
   free(end)
   free(statements1)
}

27. <statements> -> <condionalStmt> TK_END <statements1>
{
   1. STATEMENTS.list_head_syn = insert_at_head(STATEMENTS1.list_head_syn,CONDIONALSTATEMENT.node_syn);
   free(condionalStmt)
   free(end)
   free(statements1)
}

28. <statements> -> <ioStmt> TK_END <statements1>
{
   1. STATEMENTS.list_head_syn = insert_at_head(STATEMENTS1.list_head_syn,IOSTATEMENT.node_syn);
   free(ioStmt)
   free(end)
   free(statements1)
}

29. <statements> -> <declareStmt> TK_END <statements1>
{
   1. STATEMENTS.list_head_syn = insert_at_head(STATEMENTS1.list_head_syn,DECLARESTATEMENT.node_syn);
   free(declareStmt)
   free(end)
   free(statements1)
}

30. <statements> -> <simpleStmt> TK_END <statements1>
{
   1. STATEMENTS.list_head_syn = insert_at_head(STATEMENTS1.list_head_syn,SIMPLESTATEMENT.node_syn);
   free(simpleStmt)
   free(end)
   free(statements1)

}

31. <statements> -> epsilon // epsilon ko free krna hai kya
{
   STATEMENTS.list_head_syn = NULL
}

32. <ioStmt> -> TK_GET_VALUE TK_BO TK_ID TK_BC
{
   1. IOSTMT.node_syn = make_node("GET_VALUE",id.addr);
   2. free(get_value)
   3. free(bo)
   4. free(bc)
}

33. <ioStmt> -> TK_PRINT TK_BO <var_print> TK_BC
{
   1. IOSTMT.node_syn = make_node("PRINT",VAR_PRINT.node_syn);
   2. free(print)
   3. free(bo)
   4. free(VAR_PRINT)
   5. free(bc)
}

34. <boolConstt> -> TK_TRUE
{
   1. BOOLCONSTT.node_syn = true.addr;
}

35.<boolConstt> -> TK_FALSE
{
   1. BOOLCONSTT.node_syn = false.addr;
}

36. <var_print> -> TK_ID <P1>
{
   1. P1.node_inh = id.addr;
   2. VAR_print.node_syn = P1.node_syn;
   3. free(P1)
}

37. <var_print> -> TK_NUM
{
   1. VAR_print.node_syn = num.addr;
}

38. <var_print> -> TK_RNUM
{
   1. VAR_print.node_syn = rnum.addr;

}

39. <var_print> -> <boolConstt>
{
   1. VAR_print.node_syn = BOOLCONSTT.node_syn;
}

40. <P1> -> TK_SQBO <index_arr> TK_SQBC
{
   1. P1.node_syn = make_node("ARRAY_ACCESS",P1.node_inh,INDEX_ARR.node_syn);
}

41. <P1> -> epsilon
{
   1. P1.node_syn = P1.node_inh;
}

42.<simpleStmt> -> <assignmentStmt>
{
   1. SIMPLESTMT.node_syn = ASSIGNMENTSTMT.node_syn;
   2. free(ASSIGNMENTSTMT)
}

43. <simpleStmt> -> <moduleReuseStmt>
{
   1. SIMPLESTMT.node_syn = MODULEREUSESTMT.node_syn;
   2. free(MODULEREUSESTMT)
}

44. <assignmentStmt> -> TK_ID <whichStmt>
{
   1. WHICHSTMT.node_inh = id.addr
   2. ASSIGNMENTSTMT.node_syn = WHICHSTMT.node_syn
   3. free(WHICHSTMT)
}

45. <whichStmt> -> <lvalueIDStmt>
{
   1. LVALUEIDSTMT.node_inh = WHICHSTMT.node_inh
   2. WHICHSTMT.node_syn = LVALUEIDSTMT.node_syn
   3. free(LVALUEIDSTMT)
}

46. <whichStmt> -> <lvalueARRStmt>
{
   1. LVALUEARRSTMT.node_inh = WHICHSTMT.node_inh
   2. WHICHSTMT.node_syn = LVALUEARRSTMT.node_syn
   3. free(LVALUEARRSTMT)
}

47. <lvalueIDStmt> -> TK_ASSIGNOP <expression>
{
   1. LVALUEIDSTMT.node_syn = make_node("ID_ASSIGN", LVALUEIDSTMT.node_inh, EXPRESSION
.node_syn)
   2. free(assignop)
   3. free(EXPRESSION)
}

48.<lvalueARRStmt> -> TK_SQBO <element_index_with_expressions> TK_SQBC TK_ASSIGNOP <expr
ession>
{
   1. LVALUEARRSTMT.node_syn = make_node("ARRAY_ASSIGN",
                                 make_node("ARRAY_Factor", LVALUEARRSTMT.node_inh, INDEX_with_exp
ressions.node_syn),
                                 EXPRESSION.node_syn
)
   2. free(sqbo)
   3. free(INDEX_with_expressions)
   4. free(sqbc)
   5. free(assignop)
   6. free(EXPRESSION)
}

49. <index_arr> -> <sign> <new_index>
{
   1. INDEX.node_syn = make_node("index_arr", sign.node_syn,new_index.node_syn)
}

50. <new_index> -> TK_NUM
{
   1.NEW_INDEX.node_syn = num.addr
}
51. <new_index> -> TK_ID
{
   1.NEW_INDEX.node_syn = id.addr
}
52. <sign> -> TK_PLUS
{
   1.sign.node_syn = plus.addr
}
53. <sign> -> TK_MINUS
{
   1.SIGN.node_syn = minus.addr
}
54. <sign> -> epsilon
{
   1.SIGN.node_syn = NULL
}

55. MODULE_REUSE_STMT => OPTIONAL use module id with parameters ACTUAL_PARA_LIST
{

1. MODULE_REUSE_STMT.node_syn = make_node("MODULE_REUSE", ID.addr, make_node("PARAMETER_LIST1", OPTIONAL.list_head_syn), make_node("PARAMETER_LIST2", ACTUAL_PARA_LIST.list_head_syn))
2.ACTUAL_PARA_LIST.list_head_syn = NULL
3. free(OPTIONAL)
4. free(use)
5. free(module)
6. free(with)
7. free(parameters)
8. free(ACTUAL_PARA_LIST)
}

56. ACTUAL_PARA_LIST => UNARY_OP VAR_PRINT N20
{
1. ACTUAL_PARA_LIST.list_head = make_node("ACTUAL_PARA", UNARY_OP.node_syn, VAR_PRINT.node_syn)
2. N20.list_head_inh = insert_at_end(ACTUAL_PARA_LIST.list_head_inh, ACTUAL_PARA_LIST.list_head)
3. ACTUAL_PARA_LIST.list_head_syn = N20.list_head_syn
4. free(UNARY_OP)
5. free(VAR_PRINT)
6. free(N20)
}

57. ACTUAL_PARA_LIST => VAR_PRINT N20
{

1. ACTUAL_PARA_LIST.list_head = make_node("ACTUAL_PARA", VAR_PRINT.node_syn)
2. N20.list_head_inh = insert_at_end(ACTUAL_PARA_LIST.list_head_inh, ACTUAL_PARA_LIST.list_head)
2. ACTUAL_PARA_LIST.list_head_syn = N20.list_head_syn
3. free(VAR_PRINT)
4. free(N20)
}

58. N20 => comma ACTUAL_PARA_LIST
{
1. ACTUAL_PARA_LIST.list_head_inh = N20.list_head_inh
2. N20.list_head_syn = ACTUAL_PARA_LIST.list_head_syn
3. free(comma)
4. free(ACTUAL_PARA_LIST)
}

59. N20 => epsilon
{
1. N20.list_head_syn = N20.list_head_inh
}

60. OPTIONAL => sqbo IDLIST sqbc assignop
{
1. OPTIONAL.list_head_syn = IDLIST.list_head_syn
2. free(sqbo)
3. free(IDLIST)
4. free(sqbc)
5. free(assignop)

}

61. OPTIONAL => epsilon
{
   1. OPTIONAL.list_head_syn = NULL
}

62. IDLIST => id N3
{
   1. N3.list_head_inh = make_list(id)
   2. IDLIST.list_head_syn = N3.list_head_syn
   3. free(N3)
}

63. N3 => comma id N3_DASH
{
   1. N3_DASH.list_head_inh = insert_at_end(N3.list_head_inh, id)
   2. N3.list_head_syn = N3_DASH.list_head_syn
   3. free(comma)
   4. free(N3_DASH)
}

64. N3 => epsilon
{
   1. N3.list_head_syn = N3.list_head_inh
}

65. EXPRESSION => ARITHMETIC_OR_BOOL_EXPR
{
   1. EXPRESSION.node_syn = ARITHMETIC_OR_BOOL_EXPR.node_syn
   2. free(ARITHMETIC_OR_BOOL_EXPR)
}

66. EXPRESSION => U
{
   1. EXPRESSION.node_syn = U.node_syn
   2. free(U)
}

67. U => UNARY_OP NEW_NT
{
   1. U.node_syn = make_node("UNARYEXPR",UNARY_OP.node_syn, NEW_NT.node_syn)
   2. free(UNARY_OP)
   3. free(NEW_NT)
}

68. NEW_NT => bo ARITHMETIC_EXPR bc
{
   1. NEW_NT.node_syn = ARITHMETIC_EXPR.node_syn
   2. free(bo)
   3. free(bc)
   4. free(ARITHMETIC_EXPR)
}

69. NEW_NT => VAR_ID_NUM

{
1. NEW_NT.node_syn = VAR_ID_NUM.node_syn
2. free(VAR_ID_NUM)
}

70. VAR_ID_NUM => id
{
1. VAR_ID_NUM.node_syn = id.addr
}

71. VAR_ID_NUM => num
{
1. VAR_ID_NUM.node_syn = num.addr
}

72. VAR_ID_NUM => rnum
{
1. VAR_ID_NUM.node_syn = rnum.addr
}

73. UNARY_OP => plus
{
1. UNARY_OP.node_syn = plus.addr
}

74. UNARY_OP => minus
{
1. UNARY_OP.node_syn = minus.addr
}

75. ARITHMETIC_OR_BOOL_EXPR => ANY_TERM N7
{
1. N7.node_inh = ANY_TERM.node_syn
2. ARITHMETIC_OR_BOOL_EXPR.node_syn = N7.node_syn
3. free(ANY_TERM)
4. free(N7)
}
76. <N7> -> <logicalOp> <AnyTerm> <N7'>{ //maine + - konsa element hoga node me ispe dhyaan nh di
a
1. N7'.node_inh = make_node("Arithmeticop", N7.node_inh,LOGICALOP.node_syn, ANYTERM.node_
syn)
2. N7.node_syn = N7'.node_syn
3. free(LOGICALOP)
4. free(AnyTerm)
5. free(N7')
}
77.<N7> -> epsilon{
N7.node_syn = N7.node_inh
}
78 <AnyTerm> -> <arithmeticExpr> <N8>{
N8.node_inh = arithmeticExpr.node_syn
Anyterm.node_syn = N8.node_syn
free(arithmeticExpr)
free(N8)
}

79. <AnyTerm> -> <boolConstt>{
Anyterm.node_syn = boolConstt.node_syn
}
80. <N8> -> <relationalOp> <arithmeticExpr>{
N8.node_syn = make_node("Relational op",N8.node_inh,relationalOp.node_syn,arithmeticExpr.node_syn)
free(relationalOp)
free(arithmeticExpr)
}
81. <N8> -> epsilon{
N8.node_syn = N8.node_inh
}
82. <arithmeticExpr> -> <term> <N4> {
N4.node_inh = term.node_syn
arithmeticExpr.node_syn = N4.node_syn
free(term)
free(N4)
}
83. <N4> -> <op1> <term> <N4'>{
N4'.node_inh = make_node("Plus or minus",N4.node_inh,op1.node_syn,term.node_syn)
N4.node_syn = N4'.node_syn
free(op1)
free(term)
free(N4')
}

84. <N4> -> epsilon{
N4.node_syn = N4.node_inh
}
85 <term> -> <factor> <N5>{
N5.node_inh = factor.node_syn
term.node_syn = N5.node_syn
free(factor)
free(N5)
}
86. <N5> -> <op2> <factor> <N5'>{
N5'.node_inh = make_node("mulordiv",N5.node_inh,op2.node_syn,factor.node_syn)
N5.node_syn = N5'.node_syn
free(op2)
free(factor)
free(N5')
}
87. <N5> -> epsilon{
N5.node_syn = N5.node_inh
}
88. <factor> -> TK_BO <arithmeticOrBooleanExpr> TK_BC{
factor.node_syn = arithmeticOrBooleanExpr.node_syn
free(bo)
free(arithmeticOrBooleanExpr)
free(bc)
}

89. <factor> -> TK_NUM{
factor.node_syn = num.addr
}

90. <factor> -> TK_RNUM{
    factor.node_syn = rnum.addr
}

91. <factor> -> <boolConstt>{
    factor.node_syn = boolConstt.node_syn
}
92 <factor> -> TK_ID <N11>{
   N11.node_inh = id
   factor.node_syn = N11.node_syn
   free(N11)

}
93. <N11> -> TK_SQBO <element_index_with_expressions> TK_SQBC {
    N11.node_syn = make_node("arrayfactor",N11.node_inh,element_index_with_expressions.node_syn)
    free(element_index_with_expressions)
}
94. <N11> -> epsilon{
    N11.node_syn = N11.node_inh
}
95. <arrExpr> -> <arrTerm> <arr_N4>{
    arr_N4.node_inh = arrTerm.node_syn
    arrExpr.node_syn = arr_N4.node_syn
    free(arrTerm)
    free(arr_N4)
}
96. <arr_N4> -> <op1> <arrTerm> <arr_N4'>{
    arr_N4'.node_inh = make_node("plus or minus",arr_N4.node_inh,op1.node_syn,arrTerm.node_syn)
    arr_N4.node_syn = arr_N4'.node_syn
    free(op1)
    free(arrTerm)
    free(arr_N4')
}
97. <arr_N4> -> epsilon{
    arr_N4.node_syn = arr_N4.node_inh
}
98. <arrTerm> -> <arrFactor> <arr_N5>{
    arr_N5.node_inh = arrFactor.node_syn
    arrTerm.node_syn = arr_N5.node_syn
    free(arrFactor)
    free(arr_N5)
}
99. <arr_N5> -> <op2> <arrFactor> <arr_N5'>{
    arr_N5'.node_inh = make_node("mul or div",arr_N5.node_inh,op2.node_syn,arrFactor.node_syn)
    arr_N5.node_syn = arr_N5'.node_syn
    free(op2)
    free(arrFactor)
    free(arr_N5')

}
100. <arr_N5> -> epsilon{
    arr_N5.node_syn = arr_N5.node_inh
}
101. <arrFactor> -> TK_ID{

```
    arr_factor.node_syn = id.addr
}
102. <arrFactor> -> TK_NUM{
    arr_factor.mode_syn = num.addr
}
103.  <arrFactor> -> <boolConstt>{
    arr_factor.node_syn = boolConstt.node_syn
    free(bool_Constt)
}
104. <arrFactor> -> TK_BO <arrExpr> TK_BC{
    arrFactor.node_syn = arrExpr.node_syn
    free(bo)
    free(arrExpr)
    free(bc)
}
105. <element_index_with_expressions> -> <sign> <N10>{
    element_index_with_expressions.node_syn = make_node("unaryexp",sign.node_syn,N10.node_syn)
    free(sign)
    free(N10)
}
106. <element_index_with_expressions> -> <arrExpr>{
    element_index_with_expressions.node_syn = arrExpr.node_syn
    free(arrExpr)
}
107. <N10> -> <new_index>{
    N10.node_syn = new_index.node_syn
    free(new_index)
}
108. <N10> -> TK_BO <arrExpr> TK_BC{
    N10.node_syn = arrExpr.node_syn
    free(bo)
    free(arrExpr)
    free(bc)
}
109. OP1 => plus
{
    1. OP1.node_syn = plus.addr
}

110. OP1 => minus
{
    1. OP1.node_syn = minus.addr
}

111. OP2 => mul
{
    1. OP2.node_syn = mul.addr
}

112. OP2 => div
{
    1. OP2.node_syn = div.addr
}

113. LOGICAL_OP => and
```

{
1. LOGICAL_OP.node_syn = and.addr
}

114. LOGICAL_OP => or
{
1. LOGICAL_OP.node_syn = or.addr
}

115. RELATIONAL_OP => lt
{
1. RELATIONAL_OP.node_syn = lt.addr
}

116. RELATIONAL_OP => le
{
1. RELATIONAL_OP.node_syn = le.addr
}

117. RELATIONAL_OP => gt
{
1. RELATIONAL_OP.node_syn = gt.addr

}

118. RELATIONAL_OP => ge
{
1. RELATIONAL_OP.node_syn = ge.addr
}

119. RELATIONAL_OP => eq
{
1. RELATIONAL_OP.node_syn = eq.addr
}

120. RELATIONAL_OP => ne
{
1. RELATIONAL_OP.node_syn = ne.addr
}
121. <declareStmt> -> TK_DECLARE <idList> TK_COLON <dataType>{
1. DECALRESTMT.node_syn = make_node("DECLARE",DATATYPE.node_syn,IDLIST.list_head_syn)

2. free(declare)
3. free(IDLIST)
4. free(colon)
5. free(DATATYPE)
}
122. <condionalStmt> -> TK_SWITCH TK_BO TK_ID TK_BC TK_START <caseStmts> <default>{
1. CONDITIONALSTMT.node_syn = make_node("SWITCH",id.addr,
make_node("CASES", CASESTMTS.list_head_syn),
DEFAULT.node_syn
)
2. free(switch)
3. free(bo)
4. free(bc)

5. free(start)
6. free(CASESTMTS)
7. free(DEFAULTSTMT)
}
123. <caseStmts> -> TK_CASE <value> TK_COLON <statements> TK_BREAK TK_SEMICOL <N9>{
1. CASESTMT.list_head_syn = insert_at_head(N9.list_head_syn,make_node("CASE", VALUE.node_syn,
make_node("STATEMENTS", STATEMENTS.list_head_syn)
)
)
2. free(case)
3. free(VALUE)
4. free(colon)
5. free(STATEMENTS)
6. free(break)
7. free(semicol)
8. free(N9)
}
124. <N9> -> TK_CASE <value> TK_COLON <statements> TK_BREAK TK_SEMICOL <N9'>{
N9.list_head_syn = insert_at_head(N9'.list_head_syn,make_node("CASE", VALUE.node_syn,
make_node("STATEMENTS", STATEMENTS.list_head_syn)
)

2. free(case)
3. free(VALUE)
4. free(colon)
5. free(STATEMENTS)
6. free(break)
7. free(semicol)
8. free(N9')
}
125 <N9> -> epsilon{
N9.list_head_syn = NULL
}
126 <value> -> TK_NUM{
value.node_syn = num.addr
}
127 <value> -> TK_TRUE{
value.node_syn = true.addr
}
128 <value> -> TK_FALSE{
value.node_syn = false.addr
}
129. <default> -> TK_DEFAULT TK_COLON <statements> TK_BREAK TK_SEMICOL {
default.node_syn = make_node("DEFAULTCASE",make_node("STATEMENTS",STATEMENTS.list_head_syn))
free(default)
free(colon)
free(STATEMENTS)
free(break)
free(semicol)
}
130. <default> -> epsilon{

default.node_syn = NULL
}
131. <iterativeStmt> -> TK_FOR TK_BO TK_ID TK_IN <range_for_loop> TK_BC TK_START <statements>{
1. ITERATIVESTMT.node_syn = make_node("FORLOOP",id,RANGE_FOR_LOOP.node_syn,
make_node("STATEMENTS", STATEMENTS.list_head_syn)
)
2. free(for)
3. free(bo)
4. free(in)
5. free(RANGE_FOR_LOOP)
6. free(bc)
7. free(start)
8. free(STATEMENTS)

}
132. <iterativeStmt> -> TK_WHILE TK_BO <arithmeticOrBooleanExpr> TK_BC TK_START <statements>
{
ITERATIVESTMT.node_syn = make_node("WHILELOOP",ARITHMETIC_OR_BOOLEAN_EXPRESSION.node_syn,make_node("STATEMENTS",STATEMENTS.list_head_syn))
2. free(while)
3. free(bo)
4. free(ARITHMETIC_OR_BOOLEAN_EXPRESSION)
5. free(bc)
6. free(start)
7. free(STATEMENTS)
}
133. <range_for_loop> -> <index_for_loop1> TK_RANGEOP <index_for_loop2>{
RANGE_FOR_LOOP.node_syn = make_node("range_for_loop",index_for_loop1.node_syn,index_for_loop2.node_syn)
free(index_for_loop1)
free(RANGEOP)
free(index_for_loop2)
}

134. <index_for_loop> -> <sign_for_loop> <new_index_for_loop>{
INDEX_FOR_LOOP.node_syn = make_node("index_for_loop", sign_for_loop.node_syn,new_index_for_loop.node_syn)
free(sign_for_loop)
free(new_index_for_loop)
}
135. <new_index_for_loop> -> TK_NUM{
new_index_for_loop = num.addr
}
136 <sign_for_loop> -> TK_PLUS{
sign_for_loop.node_syn = plus.addr
}
137 <sign_for_loop> ->TK_MINUS{
sign_for_loop.node_syn = minus.addr
}
138 <sign_for_loop> ->epsilon{
sign_for_loop.node_syn = NULL
}