1	<program> = <moduleDeclarations> <otherModules> <driverModule> <otherModules'>	// TOP TO BOTTOM
		1. <moduleDeclarations>.inh_addr = makeNode(label:List Head)
		2. <otherModules>.inh_addr = makeNode(label:ListHead)
		// BOTTOM TO TOP
		1. <program>.addr = makeNode(label:program,<moduleDeclarations>.syn_list, <otherModules>.syn_list, <driverModule>.syn_addr, <otherModules>.syn_list)
		2. free(<moduleDeclarations>, <otherModules>, <driverModule>, <otherModules>)

2	<moduleDeclarations> = <moduleDeclaration> <moduleDeclarations>	// TOP TO BOTTOM
		<moduleDeclarations_child>.inh_addr = <moduleDeclarations>.inh_addr
		// BOTTOM TO TOP
		0. moduleDeclarationNode = makeNode(label:moduleDeclaration, <moduleDeclaration>.syn_addr)
		1. <moduleDeclarations>.syn_list = insertAtStart(<moduleDeclarations_child>.syn_list, moduleDeclarationNode)
		2. free(<moduleDeclarations_child>)
		3. free(<moduleDeclaration>)

3	<moduleDeclarations> = e	//BOTTOM TO TOP <moduleDeclarations>.syn_list = <moduleDeclarations>.inh_addr
4	<moduleDeclaration> = DECLARE MODULE ID SEMICOL	//BOTTOM TO TOP 1. <moduesDeclaration>.syn_addr = addr(ID) 2. free(DECLARE, MODULE, SEMICOL)

5	<otherModules> = <module> <otherModules>	// TOP TO BOTTOM
		<otherModules_child>.inh_addr = <otherModules>.inh_addr
		// BOTTOM TO TOP
		1. moduleNode = makeNode(label:moduleNode, <module>.syn_addr)
		2. <otherModules>.syn_list = insertAtStart(<otherModules_child>.syn_list, moduleNode)
		3. free(<otherModules_child>)
		4. free(<module>)

6	<otherModules> = e	1. <otherModules>.syn_list = <otherModules>.inh_addr
7	<driverModule> = DRIVERDEF DRIVER PROGRAM DRIVERENDDEF <moduleDef>	// BOTTOM TO TOP 1. <driverModule>.syn_addr = <moduleDef>.addr 2. free(DRIVERDEF, DRIVER, PROGRAM, DRIVERENDDEF, <moduleDef>)
8	<module> = DEF MODULE ID ENDDEF TAKES INPUT SQBO <input_plist> SQBC SEMICOL <ret	// BOTTOM TO TOP 1. <module>.syn_addr = makeNode(label:module, addr(ID), <input_plist>.syn_list, <ret>.syn_list, <moduleDef>.addr) 2. free(DEF, MODULE, ENDDEF, TAKES, INPUT, SQBO, <input_plist>, SQBC, SEMICOL, <ret>, <moduleDef>)
9	<ret> = RETURNS SQBO <output_plist> SQBC SEMICOL	// BOTTOM TO TOP 1. <ret>.syn_list = <output_plist>.syn_list 2. free(RETURNS) 3. free(SQBO) 4. free(output_plist) 5. free(SQBC) 6. free(SEMICOL)
10	<ret> = e	<ret>.syn_list = NULL
11	<input_plist> = ID COLON <dataType> <N1>	1. <input_plist>.list_head = makeNode(label:ListNode, addr(ID), <dataType>.syn_addr) 2. <N1>.inh_list = <input_plist>.list_head 3. free(COLON) 4. free(<dataType> // BOTTOM TO TOP 1. <input_plist>.syn_list = <N1>.syn_list 2. free(<N1>)
12	<N1> = COMMA ID COLON <dataType> <N1>	1. <N1>.syn_list = insertBack(<N1>.inh_list, makeNode(label:ListNode, addr(ID), <dataType>.syn_addr)) 2. free(COMMA) 3. free(COLON) 4.. free(<dataType>) 5. <N1_child>.inh_list = <N1>.syn_list // BOTTOM TO TOP 1. <N1>.syn_list = <N1_child>.syn_list 2. free(<N1_child>)
13	<N1> = e	<N1>.syn_list = <N1>.inh_list

14	<output_plist> = ID COLON <type> <N2> We do not allow <output_plist> to return array element	1. <output_plist>.list_head = makeNode(label:ListNode, addr(ID), <type>.syn_addr) 2. <N2>.inh_list = <output_plist>.list_head 3. free(COLON) 4. free(<type>) // BOTTOM TO TOP 1. <output_plist>.syn_list = <N2>.syn_list 2. free(<N2>)
15	<N2> = COMMA ID COLON <type> <N2>	1. <N2>.syn_list = insertBack(<N2>.inh_list, makeNode(label:ListNode, addr(ID), <type>.syn_addr)) 2. free(COMMA) 3. free(COLON) 4.. free(<type>) 5. <N2_child>.inh_list = <N2>.syn_list // BOTTOM TO TOP 1. <N2>.syn_list = <N2_child>.syn_list 2. free(<N2_child>)
16	<N2> = e	<N2>.syn_list = <N2>.inh_list
17	<dataType> = INTEGER	<dataType>.syn_addr = addr(INTEGER)
18	<dataType> = REAL	<dataType>.syn_addr = addr(REAL)
19	<dataType> = BOOLEAN	<dataType>.syn_addr = addr(BOOLEAN)
20	<dataType> = ARRAY SQBO <range_arrays> SQBC OF <type>	1. <dataType>.syn_addr = makeNode(label:ArrayType, <type>.syn_addr, <range_arrays>.syn_addr) 2. free(<type>) 3. free(<range_arrays>)
21	<range_arrays> = <index_arr> RANGEOP <index_arr>	1. <range_arrays>.syn_addr = makeNode(label:range_array, <index_arr>.syn_addr, <index_arr>.syn_addr) 2. free(<index_arr>) 3. free(<index_arr>) 4. free(RANGEOP)

22	<index_arr> = <sign> <new_index>	<index_arr>.syn_addr = makeNode(label:index_arr, <sign>.syn_addr, <new_index>.syn_addr)
		free(<sign>)
		free(<new_index>)

23	<new_index> = NUM	<new_index>.syn_addr = addr(NUM)
24	<new_index> = ID	<new_index>.syn_addr = addr(ID)
25	<sign> = PLUS	<sign>.syn_addr = addr(PLUS)
26	<sign> = MINUS	<sign>.syn_addr = addr(MINUS)
27	<sign> = e	<sign>.syn_addr = addr(PLUS_DEFAULT)
28	<type> = REAL	<type>.syn_addr = addr(REAL)
29	<type> = INTEGER	<type>.syn_addr = addr(INTEGER)
30	<type> = BOOLEAN	<type>.syn_addr = addr(BOOLEAN)

31	<moduleDef> = START <statements> END	// TOP TO BOTTOM
		1. statements.inh_addr = makeNode(label:List Head)
		//BOTTOM TO TOP
		<moduleDef>.syn_addr = makeNode(label: statements, <statements>.syn)

32	<statements> = <statement> <statements>	// TOP TO BOTTOM
		1. <statements_child>.inh_addr = <statements>.inh_addr
		// BOTTOM TO TOP
		1. statementNode = makeNode(label:statement, statement.syn)
		2. <statements>.syn_list = insertAtStart(<statements_child>.syn_list, statementNode)
		3. free(<statements_child>)
		4. free(<statement>)

33	<statements> = e	<statements>.syn_list = <statements>.inh_addr
34	<statement> = <ioStmt>	<statement>.syn = <ioStmt>.addr
35	<statement> = <simpleStmt>	<statement>.syn = <simpleStmt>.addr
36	<statement> = <declareStmt>	<statement>.syn = <declareStmt>.addr
37	<statement> = <conditionalStmt>	<statement>.syn = <conditionalStmt>.addr
38	<statement> = <iterativeStmt>	<statement>.syn = <iterative>.addr

39	<ioStmt> = GET_VALUE BO ID BC SEMICOL	1.<ioStmt>.addr = makeNode(label:in, ID) 2.free(GET_VALUE), free(BO), free(BC), free(SEMICOL)
40	<ioStmt> = PRINT BO <var_print> BC SEMICOL	1.<ioStmt>.addr = makeNode(label:out, <var_print>.addr) 2. free(PRINT), free(BO), free(BC), free(SEMICOL)
41	<boolConstt> = TRUE	<boolConstt>.addr = addr(TRUE)
42	<boolConstt> = FALSE	<boolConstt>.addr = addr(FALSE)
43	<id_num_rnum> = ID	<id_num_rnum>.syn_addr = addr(ID)
44	<id_num_rnum> = NUM	<id_num_rnum>.syn_addr = addr(NUM)
45	<id_num_rnum> = RNUM	<id_num_rnum>.syn_addr = addr(RNUM)
46	<var_print> = <boolConstt>	<var_print>.addr = <boolConstt>.addr
47	<var_print> = ID <P1>	<var_print>.addr = newNode(ID, <P1>.addr)
48	<var_print> = NUM	<var_print>.addr = NUM
49	<var_print> = RNUM	<var_print>.addr = RNUM
50	<P1> = SQBO <index_arr> SQBC	<P1>.addr = <index_arr>.addr
51	<P1> = e	<P1>.addr = NULL
52	<simpleStmt> = <assignmentStmt>	<simpleStmt>.addr = <assignmentStmt>.addr
53	<simpleStmt> = <moduleReuseStmt>	<simpleStmt>.addr = <moduleReuseStmt>.addr
54	<assignmentStmt> = ID <whichStmt>	<assignmentStmt>.addr = <whichStmt>.addr <whichStmt>.inh = ID
55	<whichStmt> = <lvalueIDStmt>	<whichStmt>.addr = <lvalueIDStmt>.addr <lvalueIDStmt>.inh = <whichStmt>.inh
56	<whichStmt> = <lvalueARRStmt>	<whichStmt>.addr = <lvalueARRStmt>.addr <lvalueARRStmt>.inh = <whichStmt>.inh
57	<lvalueIDStmt> = ASSIGNOP <expression> SEMICOL	<lvalueIDStmt>.addr = newNode(label:assignID, <lvalueIDStmt>.inh, <expression>.addr)
58	<lvalueARRStmt> = SQBO <element_index_with_expressions> SQBC ASSIGNOP <expression> S	<lvalueARRStmt>.addr = newNode(label:assignARR, <lvalueARRStmt>.inh, element_index_with_expressions>.addr, <expressic
59	<moduleReuseStmt> = <optional> USE MODULE ID WITH PARAMETERS <actual_para_list> SE	1. <moduleReuseStmt>.addr = newNode(label : modelReuse, <optional>.syn_addr, ID, <actual_para_list>.addr) 2. <actual_para_list>.list_head
60	<actual_para_list> = MINUS <N_13>	1. <actual_para_list>.syn = <N_13>.syn 2. <N_13>.inh = MINUS.addr
61	<N_13> = NUM <N_12>	<N_12>.inh = newNode(label:parameter, newNode(label:parameter, <N_13>.inh, RNUM.addr),) <N_13>.syn = <N_12>.syn
62	<N_13> = RNUM <N_12>	<N_12>.inh = newNode(label:parameter newNode(label:parameter, <N_13>.inh, RNUM.addr),) <N_13>.syn = <N_12>.syn
63	<N_13> = ID <N_11> <N_12>	
64	<actual_para_list> = NUM <N_12>	<actual_para_list>.syn = <N_12>.syn <N_12>.inh = NUM.addr
65	<actual_para_list> = RNUM <N_12>	<actual_para_list>.syn = <N_12>.syn <N_12>.inh = RNUM.addr
66	<actual_para_list> = <boolConstt> <N_12>	<actual_para_list>.syn = <N_12>.syn <N_12>.inh = <boolConstt>.syn
67	<actual_para_list> = ID <N_11> <N_12>	
68	<N_12> = COMMA <actual_para_list>	<actual_para_list>.inh = newNode(label:, <N_12>.inh, <actual_para_list>.syn) <N_12>.syn = <actual_para_list>.inh
69	<N_12> = e	<N_12>.syn = <N_12>.inh
70	<optional> = SQBO <idList> SQBC ASSIGNOP	*1. <optional>.syn_addr = <idList>.addr* *2. free(SQBO)* *3. free(SQBC)* *4. free(ASSIGNOP)*
71	<optional> = e	*// Do Nothing, syn_addr of <optional> is NULL*
72	<idList> = ID <N3>	1. <idList>.list_head = makeNode(label: idListNode, addr(ID)

73	<N3> = COMMA ID <N3>	1. <N3'>.inh_list = <N3>.inh_list // BOTTOM TO TOP 2. <N3>.syn_list = insertAtFront(<N3'>.syn_list, addr(ID)) 3. free(<N3'>)
74	<N3> = e	<N3>.syn_list = <N3>.inh_list
75	<expression> = <arithmeticOrBooleanExpr>	<expression>.syn = <arithmetricOrBooleanExpr>.syn
76	<expression> = <U>	<expression>.syn = <U>.syn
77	<U> = <unary_op> <new_NT>	<U>.syn = newNode(label:unaryOP, <unary_op>.syn, <new_NT>.syn)
78	<new_NT> = BO <arithmeticExpr> BC	<new_NT>.syn = <arithmeticExpr>.syn
79	<new_NT> = <id_num_rnum>	<new_NT>.syn = <id_num_rnum>.syn free(id_num_rnum)
80	<unary_op> = PLUS	<unary_op>.syn = PLUS.addr
81	<unary_op> = MINUS	<unary_op>.syn = MINUS.addr
82	<arithmeticOrBooleanExpr> = <AnyTerm> <N7>	<arithmeticOrBoolExpr>.syn = <N7>.syn <N7>.inh = <AnyTerm>.syn
83	<N7> = <logicalOp> <AnyTerm> <N7>	<N7'>.inh = newNode(label:logicalOP, logicalOP.syn, <N7>.inh, <AnyTerm>.syn) <N7>.syn = <N7'>.syn
84	<N7> = e	<N7>.inh = <N7>.syn
85	<AnyTerm> = <arithmeticExpr> <N8>	<AnyTerm>.syn = <N8>.syn <N8>.inh = <arithmeticExpr>.syn
86	<AnyTerm> = <boolConstt>	<AnyTerm>.syn = <boolConstt>.syn
87	<N8> = <relationalOp> <arithmeticExpr>	<N8>.syn = newNode(label:relationalOP, relationalOp.syn, <N8>.inh, <arithmeticExpr>.syn)
88	<N8> = e	<N8>.syn = <N8>.inh
89	<arithmeticExpr> = <term> <N4>	<arithmeticExpr>.syn = <N4>.syn <N4>.inh = <term>.syn
90	<N4> = <op1> <term> <N4>	<N4'>.inh = newNode(label:arithmeticOP, <op1>.syn, <N4>.inh, <term>.syn) <N4>.syn = <N4'>.syn
91	<N4> = e	<N4>.syn = <N4>.inh
92	<term> = <factor> <N5>	<term>.syn = <N5>.syn <N5>.inh = <factor>.syn
93	<N5> = <op2> <factor> <N5>	<N5'>.inh = newNode(label:arithmeticOP, <op2>.syn, <N5>.inh, <factor>.syn) <N5>.syn = <N5'>.syn
94	<N5> = e	<N5>.syn = <N5>.inh
95	<factor> = BO <arithmeticOrBooleanExpr> BC	<factor>.syn = <arithmeticOrBooleanExpr>.syn
96	<factor> = NUM	<factor>.syn = NUM.addr
97	<factor> = RNUM	<factor>.syn = RNUM.addr
98	<factor> = ID <N_11>	<factor>.syn = <N_11>.syn <N_11>.inh = ID
99	<N_11> = SQBO <element_index_with_expressions> SQBC	<N_11>.syn = newNode(label:array, <N_11>.inh, <element_index_with_expressions>.syn)
100	<N_11> = e	<N_11>.syn = <N_11>.inh
101	<arrExpr> = <arrTerm> <arr_N4>	<arrExpr>.syn_addr = <arr_N4>.syn_addr <arr_N4>.inh_addr = <arrTerm>.syn_addr
102	<arr_N4> = <op1> <arrTerm> <arr_N4>	<arr_N4'>.inh = newNode(label:arithmeticOP, <op1>.syn, <arr_N4>.inh, <arrTerm>.syn) <arr_N4>.syn = <arr_N4'>.syn
103	<arr_N4> = e	<arr_N4>.syn = <arr_N4>.inh
104	<arrTerm> = <arrFactor> <arr_N5>	<arrTerm>.syn = <arr_N5>.syn <arr_N5>.inh = <arrFactor>.syn
105	<arr_N5> = <op2> <arrFactor> <arr_N5>	<arr_N5'>.inh = newNode(label:arithmeticOP, <arr_N5>.inh, <arrFactor>.syn) <arr_N5>.syn = <arr_N5'>.syn
106	<arr_N5> = e	<arr_N5>.syn = <arr_N5>.inh
107	<arrFactor> = ID	<arrFactor>.syn_addr = ID.addr
108	<arrFactor> = NUM	<arrFactor>.syn_addr = NUM.addr
109	<arrFactor> = <boolConstt>	<arrFactor>.syn_addr = <boolConstt>.addr
110	<arrFactor> = BO <arrExpr> BC	<arrFactor>.syn = <arrExpr>.syn

111	<element_index_with_expressions> = <sign> <arrExpr>	<element_index_with_expressions>.syn = newNode(label:index, sign.syn, <arrExpr>.syn)
112	<op1> = PLUS	<op1>.syn_addr = addr(PLUS)
113	<op1> = MINUS	<op1>.syn_addr = addr(MINUS)
114	<op2> = MUL	<op2>.syn_addr = addr(MUL)
115	<op2> = DIV	<op2>.syn_addr = addr(DIV)
116	<logicalOp> = AND	<logicalOp>.syn_addr = addr(AND)
117	<logicalOp> = OR	<logicalOp>.syn_addr = addr(OR)
118	<relationalOp> = LT	<relationalOp>.syn_addr = addr(LT)
119	<relationalOp> = LE	<relationalOp>.addr = addr(LT)
120	<relationalOp> = GT	<relationalOp>.syn_addr = addr(GT)
121	<relationalOp> = GE	<relationalOp>.syn_addr = addr(GE)
122	<relationalOp> = EQ	<relationalOp>.syn_addr = addr(EQ)
123	<relationalOp> = NE	<relationalOp>.syn_addr = addr(NE)
124	<declareStmt> = DECLARE <idList> COLON <dataType> SEMICOL	1. <declareStmt>.addr = newNode(label:declare, <idList>.syn, <dataType>.syn) 2. free(DECLARE), free(COLON), free(SEMICOL), free(idList), free(dataType)
125	<conditionalStmt> = SWITCH BO ID BC START <caseStmts> <default> END	1. <conditionalStmt>.addr = newNode(label:switch, ID.addr, <caseStmts>.addr, <default>.addr) 2. <caseStmts>.inh = ID.syn
126	<caseStmts> = CASE <value> COLON <statements> BREAK SEMICOL <N9>	1. <caseStmts>.addr = newNode(label:case, value.addr, <statements>.addr, <N9>.addr)
127	<N9> = CASE <value> COLON <statements> BREAK SEMICOL <N9>	1. <N9>.addr = newNode(label:case, value.addr, <statements>.addr, <N9>.addr)
128	<N9> = e	1. <N9>.addr = NULL
129	<value> = NUM	1. <value>.addr = addr(NUM)
130	<value> = TRUE	1. <value>.addr = addr(TRUE)
131	<value> = FALSE	<value>.addr = addr(FALSE)
132	<default> = DEFAULT COLON <statements> BREAK SEMICOL	1. <default>.addr = makeNode(label: default, statements.addr) 2. free(DEFAULT), free(COLON), free(BREAK), free(SEMICOL)
133	<default> = e	1. <default>.addr = NULL
134	<iterativeStmt> = FOR BO ID IN <range_for_loop> BC START <statements> END	1. <iterativeStmt>.addr = makeNode(label:for, ID.addr, <range_for_loop>.syn_addr, <statements>.syn_addr) ~~2. <range_for_loop>.inh = ID.syn~~ 3. free(FOR), free(BO), free(IN), free(START), free(END)
135	<iterativeStmt> = WHILE BO <arithmeticOrBooleanExpr> BC START <statements> END	1. <iterativeStmt>.addr = makeNode(label:while, arithmeticOrBooleanExpr>.addr, statements.addr) 2. free(WHILE), free(BO), free(BC), free(START), free(END)
136	<range_for_loop> = <index_for_loop> RANGEOP <index_for_loop>	1. <range_for_loop>.addr = makeNode(label:range,<index_for_loop>.syn_addr, <index_for_loop>.syn_addr) 2. free(<index_for_loop>) 3. free(<index_for_loop>)
137	<index_for_loop> = <sign_for_loop> <new_index_for_loop>	1. <index_for_loop>.syn_addr = makeNode(label:index,<sign_for_loop>.syn_addr, <new_index_for_loop>.syn_addr) 2. free(<new_index_for_loop>) 3. free(<sign_for_loop>)
138	<new_index_for_loop> = NUM	1. <new_index_for_loop>.syn_addr = addr(NUM)
139	<sign_for_loop> = PLUS	1. <sign_for_loop>.syn_addr = addr(PLUS)
140	<sign_for_loop> = MINUS	1. <sign_for_loop>.syn_addr = addr(MINUS)
141	<sign_for_loop> = e	1. <sign_for_loop>.addr = addr(PLUS_DEFAULT) // We are defaulting to PLUS value