* program moduleDeclarations otherModules driverModule otherModules
  + program.node = make\_node ( “root” , moduleDeclarations.list , otherModules.list , driverModule.node , otherModules.list )
* moduleDeclarations moduleDeclaration moduleDeclarations
  + moduleDeclarations.list = insert\_at\_head( moduleDeclaration.node , moduleDeclarations.list )
* moduleDeclarations EPS
  + moduleDeclarations.list = NULL
* moduleDeclaration DECLARE MODULE ID SEMICOL
  + moduleDeclaration.node = make\_node (“moduleDeclaration”, newLeaf ( TK\_ID, TK\_ID.value ) )
* otherModules module otherModules
  + otherModules.list = insert\_at\_head ( module.node , otherModules.list )
* otherModules EPS
  + otherModules.list = NULL
* driverModule DRIVERDEF DRIVER PROGRAM DRIVERENDDEF moduleDef
  + driverModule.node = make\_node ( “driverModule” , moduleDef.list )
* module DEF MODULE ID ENDDEF TAKES INPUT SQBO input\_plist SQBC SEMICOL ret moduleDef
  + module.node = make\_node ( newLeaf ( TK\_ID , TK\_ID.node ) , input\_plist.list , ret.list , moduleDef.list )
* moduleDef START statements END
  + moduleDef.list = statements.list
* ret RETURNS SQBO output\_plist SQBC SEMICOL
  + ret.list = output\_plist.list
* ret EPS
  + ret.list = NULL
* input\_plist ID COLON dataType IPL
  + input\_plist.list = insert\_at\_head ( newLeaf ( TK\_ID ,TK\_ID.value ) , IPL.list )
* IPL COMMA ID COLON dataType IPL
  + IPL.list = insert\_at\_head ( newLeaf ( TK\_ID , TK\_ID.value ) , IPL.list )
* IPL EPS
  + IPL.list = NULL
* output\_plist ID COLON type OPL
  + output\_plist.list = insert\_at\_head ( newLeaf ( TKID ,TK\_ID.value ), OPL.list )
* OPL COMMA ID COLON type OPL
  + OPL.list = insert\_at\_head ( newLeaf ( TK\_ID ,TK\_ID.value ) , type.node , OPL.list )
* OPL EPS
  + OPL.list = NULL
* statements statement statements
  + statements.list = insert\_at\_head ( statement.node , statements.list )
* statements EPS
  + statements.list = NULL
* statement ioStmt
  + statement.node = ioStmt.node
* statement simpleStmt
  + statement.node = simpleStmt.node
* statement conditionalStmt
  + statement.node = conditionalStmt.node
* statement declareStmt
  + statement.node = declareStmt.node
* statement iterativeStmt
  + statement.node = iterativeStmt.node
* ioStmt GET\_VALUE BO ID BC SEMICOL
  + ioStmt.node = newLeaf ( TK\_ID , TKID.value )
* ioStmt PRINT BO print\_var BC SEMICOL
  + ioStmt.node = make\_node ( TK\_PRINT , print\_var.node )
* print\_var var
  + print\_var.node = var.node
* print\_var TRUE
  + print\_var.node = newLeaf ( TK\_BOOL , TRUE )
* print\_var FALSE
  + print\_var.node = newLeaf ( TK\_BOOL , FALSE )
* simpleStmt assignmentStmt
  + simpleStmt.node = assignmentStmt.node
* assignmentStmt ID whichStmt
  + assignmentStmt.node = make\_node ( newLeaf ( TK\_ID, TK\_ID.value ) , whichStmt.node )
* whichStmt lvalueIDStmt
  + whichStmt.node = lvalueIDStmt.node
* whichStmt lvalueARRStmt
  + whichStmt.node = lvalueARRStmt.node
* lvalueIDStmt ASSIGNOP expression\_new SEMICOL
  + lvalueIDStmt.node = expression\_new.node
* lvalueARRStmt SQBO index\_new SQBC ASSIGNOP expression\_new SEMICOL
  + lvalueARRStmt.node = make\_node( index\_new.node, expression\_new.node )
* simpleStmt moduleReuseStmt
  + simpleStmt.node = moduleReuseStmt.node
* moduleReuseStmt optional USE MODULE ID WITH PARAMETERS idList SEMICOL
  + moduleReuseStmt.node = make\_node (“moduleReuseStmt”, optional.list , newLeaf ( TK\_ID , TK\_ID.value ) , idList.list )
* optional SQBO idList SQBC ASSIGNOP
  + optional.list = idList.list
* optional EPS
  + optional.list = NULL
* declareStmt DECLARE idList COLON dataType SEMICOL
  + declareStmt.node = make\_node ( “declareStmt” , idList.list , dataType.node )
* iterativeStmt FOR BO ID IN range\_new BC START statements END
  + iterativeStmt.node = make\_node ( “for” , range\_new.node, statements.list )
* iterativeStmt WHILE BO expression BC START statements END
  + iterativeStmt.node = make\_node ( “while” , expression.node , statements.list )
* conditionalStmt SWITCH BO ID BC START CASE value COLON statements BREAK SEMICOL caseStmt default\_new END
  + conditionalStmt.node = make\_node ( newLeaf ( TK\_ID , TK\_ID.value ) , value.node , statements.list , caseStmt.list , default\_new.list )
* caseStmt CASE value COLON statements BREAK SEMICOL caseStmt
  + caseStmt.list = insert\_at\_head ( value.node , statements.list , caseStmt.list )
* caseStmt EPS
  + caseStmt.list = NULL
* default\_new DEFAULT COLON statements BREAK SEMICOL
  + default\_new.list = statements.list
* default\_new EPS
  + default\_new.list = NULL
* expression\_new expression
  + expression\_new.node = expression.node
* expression\_new U
  + expression\_new.node = U.node
* expression boolTerm bT
  + expression.node = bT.node
  + bT.inh = boolTerm.node
* bT logicalOp boolTerm bT’
  + bT’.inh = makeNode (logicalOp.node, bT.inh , boolTerm.node)
  + bT.node = bT’.node
* bT EPS
  + bt.node = bt.inh
* boolTerm arithmeticExpr aE
  + aE.inh = arithmeticExpr.node
  + boolTerm.node = aE.node
* boolTerm TRUE
  + boolTerm.node = newLeaf ( TK\_BOOL , TRUE )
* boolTerm FALSE
  + boolTerm.node = newLeaf ( TK\_BOOL , FALSE )
* aE relationalOp arithmeticExpr
  + aE.node = makeNode (relationalOp.node , aE.inh, arithmeticExpr.node)
* aE EPS
  + aE.node = aE.inh
* arithmeticExpr term aT
  + aT.inh = term.node
  + arithmeticExpr.node = aT.node
* aT pmop term aT’
  + aT’.inh = makeNode (pmop.node, aT.inh, term.node)
  + aT.node = aT’.node
* aT EPS
  + aT.node = aT.inh
* term factor aF
  + aF.inh = factor.node
  + term.node = aF.node
* aF mdop factor aF’
  + aF’.inh = makenode (mdop.node, aF.inh, factor.node)
  + aF.node = aF’.node
* aF EPS
  + aF.node = aF’.inh
* U PLUS factor\_new
  + U.node = PLUS.node
  + U.child = facor\_new.node
* U MINUS factor\_new
  + U.node = MINUS.node
  + U.child = factor\_new.node
* factor\_new BO arithmeticExpr BC
  + factor\_new.node = arithmeticExpr.node
* factor\_new var
  + factor\_new.node = var.node
* factor BO expression BC
  + factor.node=expression.node
* factor var
  + factor.node=var.node
* logicalOp AND
  + logicalOp.node=newLeaf(TK\_AND)
* logicalOp OR
  + logicalOp.node=newLeaf(TK\_OR)
* relationalOp LT
  + relationalOp.node=newLeaf(TK\_LT)
* relationalOp LE
  + relationalOp.node=newLeaf(TK\_LE)
* relationalOp GT
  + relationalOp.node=newLeaf(TK\_GT)
* relationalOp GE
  + relationalOp.node=newLeaf(TK\_GE)
* relationalOp EQ
  + relationalOp.node=newLeaf(TK\_EQ)
* relationalOp NE
  + relationalOp.node=newLeaf(TK\_NE)
* pmop PLUS
  + pmop.node=newLeaf(TK\_PLUS)
* pmop MINUS
  + pmop.node=newLeaf(TK\_MINUS)
* mdop MUL
  + mdop.node=newLeaf(TK\_MUL)
* mdop DIV
  + mdop.node=newLeaf(TK\_DIV)
* dataType INTEGER
  + dataType.node=newLeaf(TK\_INTEGER)
* dataType REAL
  + dataType.node=newLeaf(TK\_REAL)
* dataType BOOLEAN
  + dataType.node=newLeaf(TK\_BOOLEAN)
* dataType ARRAY SQBO range SQBC OF type
  + dataType.node = makeNode(range.node , type.node)
* type INTEGER
  + type.node=newLeaf(TK\_INTEGER)
* type REAL
  + type.node=newLeaf(TK\_REAL)
* type BOOLEAN
  + type.node=newLeaf(TK\_BOOLEAN)
* var ID whichID
  + var.node = makeNode(newLeaf(TK\_ID) , whichId.node)
* var NUM
  + var.node = newLeaf((NUM , NUM.value))
* var RNUM
  + var.node= newLeaf(RNUM , RNUM.value)
* whichID SQBO index\_new SQBC
  + whichID.node = index\_new.node
* whichID EPS
  + whichID.node=NULL
* index\_new NUM
  + index\_new.node = newLeaf(NUM , NUM.value)
* index\_new ID
  + index\_new.node = newLeaf(TK\_ID, ID\_value)
* value NUM
  + value.node = newLeaf ( TK\_NUM , NUM.value )
* value TRUE
  + value.node = newLeaf ( TK\_BOOL , TRUE )
* value FALSE
  + value.node = newLeaf ( TK\_BOOL , FALSE )
* range index\_new1 RANGEOP index\_new2
  + range.node = makeNode ( “range” , index\_new1.node , index\_new2.node)
* idList ID idL
  + idList.list  = insert\_at\_head ( “TK\_ID” , idL.list )
* idL COMMA ID idL
  + idL.list = makeNode ( “TK\_ID” , idL.list )
* idL EPS
  + idL.list =  NULL
* range\_new NUM RANGEOP NUM
  + range\_new.node = makeNode( newLeaf(TK\_NUM,NUM1.value) , newLeaf(TK\_NUM,NUM2.value) )