Group - 11
Lavanya Soni (2017A7PS0151P)
Kushagra Agrawal (2017A7PS0107P)
Siddhant Khandelwal (2017A7PS0127P)

The semantic rule for each grammar rule follows it.
The semantic rule is *italicised.*

BEGIN

program moduledeclarations othermodules drivermodule othermodules
program.node = new Node(moduleDeclarations.node, otherModules.node, driverModule.node, otherModules.node)

---

moduledeclarations moduledeclaration moduledeclarations
moduleDeclarations.node = new Node(moduleDeclaration.node, moduleDeclarations.node)

moduledeclarations E
moduleDeclarations.node = NULL

---

moduledeclaration DECLARE MODULE ID SEMICOL
moduleDeclaration.node = ID.entry

---

othermodules module othermodules
otherModules.node = new Node(module.node, otherModules.node)

othermodules E
otherModules.node = NULL

---

drivermodule DRIVERDEF DRIVER PROGRAM DRIVERENDDEF moduledef
driverModule.node = moduleDef.node

---

module DEF MODULE ID ENDDEF TAKES INPUT SQBO inputplist SQBC SEMICOL ret moduledef
module.node = new Node(ID.entry , inputplist.node, ret.node, moduleDef.node)

---

ret RETURNS SQBO outputplist SQBC SEMICOL
ret.node = output_plist.node

ret E

ret.node = NULL

---

inputplist ID COLON datatype inputplistnew
inputplist.node = new Node(ID.entry , datatype.node, inputplistnew.node)

---

inputplistnew COMMA ID COLON datatype inputplistnew1
inputplistnew.node = new Node(ID.entry , datatype.node, inputplistnew1.node)

inputplistnew E
inputplistnew.node = NULL

---

outputplist ID COLON type outputplistnew
outputplist.node = new Node(ID.entry, datatype.node, outputplistnew.node)

---

outputplistnew COMMA ID COLON type outputplistnew1
outputplistnew.node = new Node(ID.entry, datatype.node, outputplistnew1.node)

outputplistnew E
outputplistnew.node = NULL

---

datatype INTEGER
datatype.node = new Leaf(INTEGER)

datatype REAL
datatype.node = new Leaf(REAL)

datatype BOOLEAN
datatype.node = new Leaf(BOOLEAN)

datatype ARRAY SQBO rangearrays SQBC OF type
datatype.node = new Node(rangearrays.node, type.node)

---

type INTEGER
type.node = new Leaf(INTEGER)

type REAL
type.node = new Leaf(REAL)

type BOOLEAN
type.node = new Leaf(BOOLEAN)

---

moduledef START statements END
moduledef.node = statements.node

---

statements statement statements1
statements.node = new Node(statement.node, statements1.node)

statements E
statements.node = NULL

---

statement iostmt
statement.node = iostmt.node

statement simplestmt
statement.node = simplestmt.node

statement declarestmt
statement.node = declarestmt.node

statement conditionalstmt
statement.node =conditionalstmt.node

statement iterativestmt
statement.node = itertativestmt.node

---

iostmt GET_VALUE BO ID BC SEMICOL
iostmt.node = new Node(GET_VALUE, ID.entry)

iostmt PRINT BO var BC SEMICOL
iostmt.node = new Node(PRINT, var.node)

---

var varidnum
var.node = varidnum.node

var boolconstt
var.node = boolconstt.node

---

boolconstt TRUE
boolconstt.node = new Leaf(TRUE)

boolconstt FALSE
boolconstt.node = new Leaf(FALSE)

---

varidnum ID whichid
varidnum.node = new Node(ID.entry, whichid.node)

varidnum NUM
varidnum.node = new Leaf(NUM)

varidnum RNUM
varidnum.node = new Leaf(RNUM)

---

whichid SQBO index SQBC
whichid.node = index.node

whichid E
whichid.node = NULL

---

simplestmt assignmentstmt
simplestmt.node = assignmentstmt.node

simplestmt modulereusestmt
simplestmt.node = modulereusestmt.node

---

assignmentstmt ID whichstmt
whichstmt.inh = ID.entry
assignmentstmt.node = whchstmt.node

---

whichstmt lvalueidstmt
lvalueidstmt.inh = whichstmt.inh
whichstmt.node = lvalueidstmt.node

whichstmt lvaluearrstmt
lvaluearrstmt.inh = whichstmt.inh
whichstmt.node = lvaluearrstmt.node

---

lvalueidstmt ASSIGNOP expression SEMICOL
lvalueidstmt.node = new Node(“ASSIGNOP”, lvalueidtstmt.inh, expression.node)

---

lvaluearrstmt SQBO index SQBC ASSIGNOP expression SEMICOL
lvaluearrstmt.node = new Node(“ASSIGNOP”, index.node, expression.node)
index.inh = lavluearrstmt.inh

---

index NUM
index.node = new Node(“ARRAY_ELEMENT”, index.inh, ,NUM.val)

index ID
index.node = new Node(“ARRAY_ELEMENT”, index.inh ,ID.entry)

---

modulereusestmt optional USE MODULE ID WITH PARAMETERS idlist SEMICOL
optional.inh = idlist.node
modulereusestmt.node = optional.node

---

optional SQBO idlist SQBC ASSIGNOP
optional.node = new Node(“ASSIGNOP”, optional.inh, idlist.node)

optional E
optional.node = optional.inh

---

idlist ID idlistnew
idlist.node = new Node(ID.entry, idlistnew.node)

---

idlistnew COMMA ID idlistnew1
idlistnew.node = new Node(ID.entry , idlistnew1.node)

idlistnew E
idlistnew.node = NULL

---

expression arithmeticorbooleanexpression
expression.node = arithmeticorbooleanexpression.node

expression u
expression.node = u.node

---

u unaryop newnt
u.node = new Node(unaryop.node, newnt.node)

---

newnt BO arithmeticexpr BC
newnt.node = arithemeticexpr.node

newnt varidnum
newnt.node = varidnum.node

---

unaryop PLUS
unaryop.node = new Leaf(PLUS)

unaryop MINUS
unaryop.node = new Leaf(MINUS)

---

arithmeticorbooleanexpression anyterm n7
arithemeticorbooleanexpr.node = n7.syn
n7.inh = anyterm.node

---

n7 logicalop anyterm n7_1
n7_1.inh = new Node(logicalop.node, n7.inh, anyterm.node)
n7.syn = n7_1.syn

n7 E
n7.syn = n7.inh

---

anyterm arithmeticexpr n8
anyterm.node = new Node(aritheticexpr.node, n8.node)

anyterm boolconstt
anyterm.node = boolconstt.node

---

n8 relationalop arithmeticexpr
n8.node = new Node(relationalop.node, arithemeticexpr.node)

n8 E
n8.node = NULL

---

arithmeticexpr term n4
arithemeticexpr.node = n4.syn
n4.inh = term.node

---

n4 op1 term n4_1
n4_1.inh = new Node(op1.node, n4.inh, term.node)
n4.syn = n4_1.syn

n4 E
n4.syn = n4.inh

---

term factor n5
term.node = n5.syn
n5.inh = factor.node

---

n5 op2 factor n5_1
n5_1.inh = new Node(op2.node, n5.inh,factor.node)
n5.syn = n5_1.syn

n5 E
n5.syn = n5.inh

---

factor BO arithmeticorbooleanexpression BC
factor.node = arithemeticorbooleanexpression.node

factor varidnum
factor.node = varidnum.node

---

op1 PLUS
op1.node = new Leaf(PLUS)

op1 MINUS
op1.node = new Leaf(MINUS)

---

op2 MUL
op2.node = new Leaf(MUL)

op2 DIV
op2.node= new Leaf(DIV)

---

logicalop AND
logicalop.node = new Leaf(AND)

logicalop OR

logicalop.node = new Leaf(OR)

---

relationalop LT
relationalop.node = new Leaf(LT)

relationalop LE
relationalop.node = new Leaf(LE)

relationalop GT
relationalop.node = new Leaf(GT)

relationalop GE
relationalop.node = new Leaf(GE)

relationalop EQ
relationalop.node = new Leaf(EQ)

relationalop NE
relationalop.node = new Leaf(NE)

---

declarestmt DECLARE idlist COLON datatype SEMICOL
declarestmt.node = new Node(idlist.node , datatype.node)

---

conditionalstmt SWITCH BO ID BC START casestmts default END
conditionalstmt.node = new Node (ID.entry , casestmts.node, default.node)

---

casestmts CASE value COLON statements BREAK SEMICOL n9
casestmts.node = new Node(value.node , statements.node , n9.node)

---

n9 CASE value COLON statements BREAK SEMICOL n9_1
n9.node = new Node(value.node , statements.node , n9_1.node)

n9 E
n9.node = NULL

---

value NUM
value.node = new Leaf(NUM, NUM.val)

value TRUE
value.node = new Leaf(TRUE)

value FALSE
value.node = new Leaf(FALSE)

---

default DEFAULT COLON statements BREAK SEMICOL
default.node = new Node(statements.node)

default E
default.node = NULL

---

iterativestmt FOR BO ID IN range BC START statements END
iterativestmt.node = new Node(ID.entry , range.node, statements.node)

iterativestmt WHILE BO arithmeticorbooleanexpression BC START statements END
iterativestmt.node = new Node(artithemeticorbooleanexpression.node, statements.node)

---

range NUM RANGEOP NUM
range.node = new Node(new Leaf(NUM, NUM.val), new Leaf(NUM, NUM.val))

---

rangearrays index RANGEOP index
rangearrays.node = new Node(index.node, index.node)

END