```
/* GROUP 51
Arnav Agarwal:                  2019B2A70966P
Aviral Omar:                    2019B3A70411P
Chandra Sekhar Reddy E:         2019B4A70634P
Vatsal Pattani:                 2019B5A70697P
*/
1       program => moduleDeclarations otherModules1 driverModule otherModules2
{
        Bottom-Up:
        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))
        free(moduleDeclarations)
        free(otherModules1)
        free(driverModule)
        free(otherModules2)
}
2       moduleDeclarations => moduleDeclaration moduleDeclarations1
{
        Bottom-Up:
        moduleDeclarations.list_head_syn = insert_at_head(moduleDeclarations1.list_head_syn,
moduleDeclaration.node_syn)
        free(moduleDeclaration)
        free(moduleDeclarations1)
}
3       moduleDeclarations => epsilon
{
        Bottom-Up:
        moduleDeclarations.list_head_syn = NULL
        free(epsilon)
}
4       moduleDeclaration => DECLARE MODULE ID SEMICOL
{
        Bottom-Up:
        moduleDeclaration.node_syn = make_node(ID)
        free(DECLARE)
        free(MODULE)
        free(ID)
        free(SEMICOL)
}
5       otherModules => module otherModules1
{
        Bottom-Up:
        otherModules.list_head_syn = insert_at_head(otherModules1.list_head_syn,
module.node_syn)
        free(module)
        free(otherModules1)
}
6       otherModules => epsilon
```

```
{
    Bottom-Up:
    otherModules.list_head_syn = NULL
    free(epsilon)
}
7   driverModule => DRIVERDEF DRIVER PROGRAM DRIVERENDDEF moduleDef
{
    Bottom-Up:
    driverModule.node_syn = make_node("Driver", moduleDef.node_syn)
    free(DRIVERDEF)
    free(DRIVER)
    free(PROGRAM)
    free(DRIVERENDDEF)
    free(moduleDef)
}
8   module => DEF MODULE ID ENDDEF TAKES INPUT SQBO inputPList SQBC SEMICOL ret moduleDef
{
    Bottom-Up:
    module.node_syn = make_node("Module", make_node(ID), make_node("ParametersList", inputPList.list_head_syn), ret.node_syn, moduleDef.node_syn)
    free(DEF)
    free(MODULE)
    free(ID)
    free(ENDDEF)
    free(TAKES)
    free(INPUT)
    free(SQBO)
    free(inputPList)
    free(SQBC)
    free(SEMICOL)
    free(ret)
    free(moduleDef)
}
9   ret => RETURNS SQBO outputPList SQBC SEMICOL
{
    Bottom-Up:
    ret.node_syn = make_node("ParametersList", outputPList.list_head_syn)
    free(RETURNS)
    free(SQBO)
    free(outputPList)
    free(SQBC)
    free(SEMICOL)
}
10  ret => epsilon
{
    Bottom-Up:
    ret.node_syn = make_node("ParametersList", NULL)
    free(epsilon)
}
```

```
11	inputPList => ID COLON dataType iPList2
{
	Bottom-Up:
	inputPList.list_head_syn = insert_at_head(iPList2.list_head_syn, make_node("InputVarType", make_node(ID), dataType.node_syn))
	free(ID)
	free(COLON)
	free(dataType)
	free(iPList2)
}
12	iPList2 => COMMA ID COLON dataType iPList21
{
	Bottom-Up:
	iPList21.list_head_inh = insert_at_head(iPList21.list_head_syn, make_node("InputVarType", make_node(ID), dataType.node_syn))
	free(COMMA)
	free(ID)
	free(COLON)
	free(dataType)
	free(iPList21)
}
13	iPList2 => epsilon
{
	Bottom-Up:
	iPList2.list_head_syn = NULL
	free(epsilon)
}
14	outputPList => ID COLON type oPList2
{
	Bottom-Up:
	outputPList.list_head_syn = insert_at_head(oPList2.list_head_syn, make_node("OutputVarType", make_node(ID), type.node_syn))
	free(ID)
	free(COLON)
	free(type)
	free(oPList2)
}
15	oPList2 => COMMA ID COLON type oPList21
{
	Bottom-Up:
	outputPList.list_head_syn = insert_at_head(oPList2.list_head_syn, make_node("OutputVarType", make_node(ID), type.node_syn))
	free(COMMA)
	free(ID)
	free(COLON)
	free(type)
	free(oPList21)
}
16	oPList2 => epsilon
{
        Bottom-Up:
        oPList2.list_head_syn = NULL
        free(epsilon)
}
17      dataType => INTEGER
{
        Bottom-Up:
        dataType.node_syn = make_node(INTEGER)
        free(INTEGER)
}
18      dataType => REAL
{
        Bottom-Up:
        dataType.node_syn = make_node(REAL)
        free(REAL)
}
19      dataType => BOOLEAN
{
        Bottom-Up:
        dataType.node_syn = make_node(BOOLEAN)
        free(BOOLEAN)
}
20      dataType => ARRAY SQBO arrRange SQBC OF type
{
        Bottom-Up:
        dataType.node_syn = make_node("Array", arrRange.node_syn, type.node_syn)
        free(ARRAY)
        free(SQBO)
        free(arrRange)
        free(SQBC)
        free(OF)
        free(type)
}
21      arrRange => signedIndex1 RANGEOP signedIndex2
{
        Bottom-Up:
        arrRange.node_syn =
make_node("ArrayRange",signedIndex1.node_syn,signedIndex2.node_syn)
        free(signedIndex1)
        free(rangeop)
        free(signedIndex2)
}
22      type => INTEGER
{
        Bottom-Up:
        type.node_syn = make_node(INTEGER)
        free(INTEGER)
}
23      type => REAL
{
        Bottom-Up:
        type.node_syn = make_node(REAL)
        free(REAL)
}
24      type => BOOLEAN
{
        Bottom-Up:
        type.node_syn = make_node(BOOLEAN)
        free(BOOLEAN)
}
25      moduleDef => START statements END
{
        Bottom-Up:
        moduleDef.node_syn = make_node("Statements", statements.list_head_syn)
        free(START)
        free(statements)
        free(END)
}
26      statements => statement statements1
{
        Bottom-Up:
        statements.list_head_syn = insert_at_head(statements1.list_head_syn, statement.node_syn)
        free(statement)
        free(statements1)
}
27      statements => epsilon
{
        Bottom-Up:
        statements.list_head_syn = NULL
        free(epsilon)
}
28      statement => ioStmt
{
        Bottom-Up:
        statement.node_syn = ioStmt.node_syn
        free(ioStmt)
}
29      statement => simpleStmt
{
        Bottom-Up:
        statement.node_syn = simpleStmt.node_syn
        free(simpleStmt)
}
30      statement => declareStmt
{
        Bottom-Up:
        statement.node_syn = declareStmt.node_syn
        free(declareStmt)
}
31      statement => conditionalStmt
```

```
{
        Bottom-Up:
        statement.node_syn = conditionalStmt.node_syn
        free(conditionalStmt)
}
32      statement => iterativeStmt
{
        Bottom-Up:
        statement.node_syn = iterativeStmt.node_syn
        free(iterativeStmt)
}
33      ioStmt => GET_VALUE BO ID BC SEMICOL
{
        Bottom-Up:
        ioStmt.node_syn = make_node("GetValue", make_node(ID))
        free(GET_VALUE)
        free(BO)
        free(ID)
        free(BC)
        free(SEMICOL)
}
34      ioStmt => PRINT BO varPrint BC SEMICOL
{
        Bottom-Up:
        ioStmt.node_syn = make_node("Print",varPrint.node_syn)
        free(PRINT)
        free(BO)
        free(varPrint)
        free(BC)
        free(SEMICOL)
}
35      varPrint => ID arrIndex
{
        Top-Down:
        arrIndex.node_inh = ID

        Bottom-Up:
        varPrint.node_syn = arrIndex.node_syn
        free(arrIndex)
}
36      varPrint => NUM
{
        Bottom-Up:
        varPrint.node_syn = make_node(NUM)
        free(NUM)
}
37      varPrint => RNUM
{
        Bottom-Up:
        varPrint.node_syn = make_node(RNUM)
        free(RNUM)
}
38      varPrint => boolConst
{
        Bottom-Up:
        varPrint.node_syn = boolConst.node_syn
        free(boolConst)
}
39      boolConst => TRUE
{
        Bottom-Up:
        boolConst.node_syn = make_node(TRUE)
        free(TRUE)
}
40      boolConst => FALSE
{
        Bottom-Up:
        boolConst.node_syn = make_node(FALSE)
        free(FALSE)
}
41      arrIndex => SQBO signedIndex SQBC
{
        Bottom-Up:
        arrIndex.node_syn = make_node("ArrayAccess", arrIndex.node_inh, signedIndex.node_syn);
        free(SQBO)
        free(signedIndex)
        free(SQBC)
}
42      arrIndex => epsilon
{
        Bottom-Up:
        arrIndex.node_syn = arrIndex.node_inh;
        free(epsilon)
}
43      simpleStmt => moduleReuseStmt
{
        Bottom-Up:
        simpleStmt.node_syn = moduleReuseStmt.node_syn;
        free(moduleReuseStmt)
}
44      simpleStmt => assignmentStmt
{
        Bottom-Up:
        simpleStmt.node_syn = assignmentStmt.node_syn;
        free(assignmentStmt)
}
45      assignmentStmt => ID whichStmt
{
        Top-Down:
        whichStmt.node_inh = ID
        Bottom-Up:
        assignmentStmt.node_syn = whichStmt.node_syn
        free(whichStmt)
}
46      whichStmt => lValueIDStmt
{
        Top-Down:
        lValueIDStmt.node_inh = whichStmt.node_inh

        Bottom-Up:
        whichStmt.node_syn = lValueIDStmt.node_syn
        free(lValueIDStmt)
}
47      whichStmt => lValueArrStmt
{
        Top-Down:
        lValueArrStmt.node_inh = whichStmt.node_inh

        Bottom-Up:
        whichStmt.node_syn = lValueArrStmt.node_syn
        free(lValueArrStmt)
}
48      lValueIDStmt => ASSIGNOP expression SEMICOL
{
        Bottom-Up:
        lValueIDStmt.node_syn = make_node("Assign", lValueIDStmt.node_inh,
expression.node_syn)
        free(ASSIGNOP)
        free(expression)
        free(SEMICOL)
}
49      lValueArrStmt => SQBO indexWithExpressions SQBC ASSIGNOP expression SEMICOL
{
        Bottom-Up:
        lValueArrStmt.node_syn = make_node("ArrayAssign", make_node("ArrayAccess",
lValueArrStmt.node_inh, indexWithExpressions.node_syn), expression.node_syn)
        free(SQBO)
        free(indexWithExpressions)
        free(SQBC)
        free(ASSIGNOP)
        free(expression)
        free(SEMICOL)
}
50      signedIndex => sign index
{
        Bottom-Up:
        signedIndex.node_syn = make_node("SignedIndex", sign.node_syn, index.node_syn)
        free(sign)
        free(index)
```

```
}
51      index => NUM
{
        Bottom-Up:
        index.node_syn = make_node(NUM)
        free(NUM)
}
52      index => ID
{
        Bottom-Up:
        index.node_syn = make_node(ID)
        free(ID)
}
53      sign => PLUS
{
        Bottom-Up:
        sign.node_syn = make_node(PLUS)
        free(PLUS)
}
54      sign => MINUS
{
        Bottom-Up:
        sign.node_syn = make_node(MINUS)
        free(MINUS)
}
55      sign => epsilon
{
        Bottom-Up:
        sign.node_syn = NULL
        free(epsilon)
}
56      moduleReuseStmt => optional USE MODULE ID WITH PARAMETERS actualPList SEMICOL
{
        Bottom-Up:
        moduleReuseStmt.node_syn = make_node("FunctionCall", make_node("VariableList",
optional.list_head_syn), make_node(ID), make_node("ActualParametersList",
actualPList.list_head_syn))
        free(optional)
        free(USE)
        free(MODULE)
        free(ID)
        free(WITH)
        free(PARAMETERS)
        free(actualPList)
        free(SEMICOL)
}
57      actualPList => sign param actualPList2
{
        Bottom-Up:
```

```
        actualPList.list_head_syn = insert_at_head(actualPList2.list_head_syn,
make_node("SignedParam", sign.node_syn, param.node_syn))
        free(sign)
        free(param)
        free(actualPList2)
}
58      actualPList2 => COMMA sign param actualPList21
{
        Bottom-Up:
        actualPList.list_head_syn = insert_at_head(actualPList21.list_head_syn,
make_node("SignedParam", sign.node_syn, param.node_syn))
        free(COMMA)
        free(sign)
        free(param)
        free(actualPList 21)
}
59      actualPList2 => epsilon
{
        Bottom-Up:
        actualPList2.list_head_syn = NULL
        free(epsilon)
}
60      param => NUM
{
        Bottom-Up:
        param.node_syn = make_node(NUM)
        free(NUM)
}
61      param => RNUM
{
        Bottom-Up:
        param.node_syn = make_node(RNUM)
        free(RNUM)
}
62      param => boolConst
{
        Bottom-Up:
        param.node_syn = boolConst.node_syn
        free(boolConst)
}
63      param => ID arrIndexWithExpressions
{
        Top-Down:
        arrIndexWithExpressions.node_inh = ID

        Bottom-Up:
        param.node_syn = arrIndexWithExpressions.node_syn
        free(ID)
        free(arrIndexWithExpressions)
}
```

```
64      optional => SQBO idList SQBC ASSIGNOP
{
        Bottom-Up:
        optional.list_head_syn = idList.list_head_syn
        free(SQBO)
        free(idList)
        free(SQBC)
        free(ASSIGNOP)
}
65      optional => epsilon
{
        Bottom-Up:
        optional.list_head_syn = NULL
        free(epsilon)
}
66      idList => ID idList2
{
        Bottom-Up:
        idList.list_head_syn = insert_at_head(idList2.list_head_syn, make_list(ID))
        free(ID)
        free(idList2)
}
67      idList2 => COMMA ID idList21
{
        Bottom-Up:
        idList.list_head_syn = insert_at_head(idList21.list_head_syn, make_list(ID))
        free(COMMA)
        free(ID)
        free(idList2)
}
68      idList2 => epsilon
{
        Bottom-Up:
        idList2.list_head_syn = NULL
        free(epsilon)
}
69      expression => arithmeticOrLogicalExpr
{
        Bottom-Up:
        expression.node_syn = arithmeticOrLogicalExpr.node_syn
        free(arithmeticOrLogicalExpr)
}
70      expression => unaryOpExpr
{
        Bottom-Up:
        expression.node_syn = unaryOpExpr.node_syn
        free(unaryOpExpr)
}
71      unaryOpExpr => unaryOp unsignedArithExpr
{
        Bottom-Up:
        unaryOpExpr.node_syn = make_node("UnaryOpExpr", unaryOp.node_syn,
unsignedArithExpr.node_syn)
        free(unaryOp)
        free(unsignedArithExpr)
}
72      unsignedArithExpr => BO arithmeticExpr BC
{
        Bottom-Up:
        unsignedArithExpr.node_syn = arithmeticExpr.node_syn
        free(BO)
        free(arithmeticExpr)
        free(BC)
}
73      unsignedArithExpr => varIDNum
{
        Bottom-Up:
        unsignedArithExpr.node_syn = varIDNum.node_syn
        free(varIDNum)
}
74      unaryOp => PLUS
{
        Bottom-Up:
        unaryOp.node_syn = make_node(PLUS)
        free(PLUS)
}
75      unaryOp => MINUS
{
        Bottom-Up:
        unaryOp.node_syn = make_node(MINUS)
        free(MINUS)
}
76      varIDNum => ID
{
        Bottom-Up:
        varIDNum.node_syn = make_node(ID)
        free(ID)
}
77      varIDNum => NUM
{
        Bottom-Up:
        varIDNum.node_syn = make_node(NUM)
        free(NUM)
}
78      varIDNum => RNUM
{
        Bottom-Up:
        varIDNum.node_syn = make_node(RNUM)
        free(RNUM)
}
```

```
79      arithmeticOrLogicalExpr => anyTerm logicalOpExpr
{
        Top-Down:
        logicalOpExpr.node_inh = anyTerm

        Bottom-Up:
        arithmeticOrLogicalExpr.node_syn = logicalOpExpr.node_syn
        free(logicalOpExpr)
}
80      logicalOpExpr => logicalOp anyTerm logicalOpExpr1
{
        Top-Down:
        logicalOpExpr1.node_inh = anyTerm

        Bottom-Up:
        logicalOpExpr.node_syn = make_node(logicalOp.node_syn, logicalOpExpr.node_inh, logicalOpExpr1.node_syn)
        free(logicalOp)
        free(logicalOpExpr1)
}
81      logicalOpExpr => epsilon
{
        Bottom-Up:
        logicalOpExpr.node_syn = logicalOpExpr.node_inh
        free(epsilon)
}
82      anyTerm => arithmeticExpr relationOpExpr
{
        Top-Down:
        relationOpExpr.node_inh = arithmeticExpr

        Bottom-Up:
        anyTerm.node_syn = relationOpExpr.node_syn
        free(relationOpExpr)
}
83      relationOpExpr => relationalOp arithmeticExpr
{
        Bottom-Up:
        relationOpExpr.node_syn = make_node(relationalOp.node_syn, relationOpExpr.node_inh, arithmeticExpr.node_syn)
        free(relationalOp)
        free(arithmeticExpr)
}
84      relationOpExpr => epsilon
{
        Bottom-Up:
        relationOpExpr.node_syn = relationOpExpr.node_inh
        free(epsilon)
}
85      arithmeticExpr => term addSubExpr
```

```
{
            Top-Down:
            addSubExpr.node_inh = term

            Bottom-Up:
            arithmeticExpr.node_syn = addSubExpr.node_syn
            free(addSubExpr)
}
86          addSubExpr => addSubOp term addSubExpr1
{
            Top-Down:
            addSubExpr1.node_inh = term

            Bottom-Up:
            addSubExpr.node_syn = make_node(addSubOp.node_syn, addSubExpr.node_inh,
addSubExpr1.node_syn)
            free(addSubOp)
            free(addSubExpr1)
}
87          addSubExpr => epsilon
{
            Bottom-Up:
            addSubExpr.node_syn = addSubExpr.node_inh
            free(epsilon)
}
88          term => factor mulDivExpr
{
            Top-Down:
            mulDivExpr.node_inh = factor

            Bottom-Up:
            term.node_syn = mulDivExpr.node_syn
            free(mulDivExpr)
}
89          mulDivExpr => mulDivOp factor mulDivExpr1
{
            Top-Down:
            mulDivExpr1.node_inh = factor

            Bottom-Up:
            mulDivExpr.node_syn = make_node(mulDivOp.node_syn, mulDivExpr.node_inh,
mulDivExpr1.node_syn)
            free(mulDivOp)
            free(mulDivExpr1)
}
90          mulDivExpr => epsilon
{
            Bottom-Up:
            mulDivExpr.node_syn = mulDivExpr.node_inh
            free(epsilon)
```

```
}
91      factor => BO arithmeticOrLogicalExpr BC
{
        Bottom-Up:
        factor.node_syn = arithmeticOrLogicalExpr.node_syn
        free(BO)
        free(arithmeticOrLogicalExpr)
        free(BC)
}
92      factor => NUM
{
        Bottom-Up:
        factor.node_syn = make_node(NUM)
        free(NUM)
}
93      factor => RNUM
{
        Bottom-Up:
        factor.node_syn = make_node(RNUM)
        free(RNUM)
}
94      factor => boolConst
{
        Bottom-Up:
        factor.node_syn = boolConst.node_syn
        free(boolConst)
}
95      factor => ID arrIndexWithExpressions
{
        Top-Down:
        arrIndexWithExpressions.node_inh = ID

        Bottom-Up:
        factor.node_syn = arrIndexWithExpressions.node_syn
        free(ID)
        free(arrIndexWithExpressions)
}
96      arrIndexWithExpressions => SQBO indexWithExpressions SQBC
{
        Bottom-Up:
        arrIndexWithExpressions.node_syn = make_node("ArrayAccess",
arrIndexWithExpressions.node_inh, indexWithExpressions.node_syn)
        free(SQBO)
        free(indexWithExpressions)
        free(SQBC)
}
97      arrIndexWithExpressions => epsilon
{
        Bottom-Up:
        arrIndexWithExpressions.node_syn = arrIndexWithExpressions.node_inh
        free(epsilon)
}
98      indexWithExpressions => sign arrExpr
{
        Bottom-Up:
        indexWithExpressions.node_syn = make_node("IndexWithExpressions", sign.node_syn,
arrExpr.node_syn)
        free(sign)
        free(arrExpr)
}
99      arrExpr => arrTerm arrAddSubExpr
{
        Top-Down:
        arrAddSubExpr.node_inh = arrTerm

        Bottom-Up:
        arrExpr.node_syn = arrAddSubExpr.node_syn
        free(arrAddSubExpr)
}
100     arrAddSubExpr => addSubOp arrTerm arrAddSubExpr1
{
        Top-Down:
        arrAddSubExpr1.node_inh = arrTerm

        Bottom-Up:
        arrAddSubExpr.node_syn = make_node(addSubOp.node_syn, arrAddSubExpr.node_inh,
arrAddSubExpr1.node_syn)
        free(addSubOp)
        free(arrAddSubExpr1)
}
101     arrAddSubExpr => epsilon
{
        Bottom-Up:
        arrAddSubExpr.node_syn = arrAddSubExpr.node_inh
        free(epsilon)
}
102     arrTerm => arrFactor arrMulDivExpr
{
        Top-Down:
        arrMulDivExpr.node_inh = arrFactor

        Bottom-Up:
        arrTerm.node_syn = arrMulDivExpr.node_syn
        free(arrMulDivExpr)
}
103     arrMulDivExpr => mulDivOp arrFactor arrMulDivExpr1
{
        Top-Down:
        arrMulDivExpr1.node_inh = arrFactor
        Bottom-Up:
        arrMulDivExpr.node_syn = make_node(mulDivOp.node_syn, arrMulDivExpr.node_inh,
arrMulDivExpr1.node_syn)
        free(mulDivOp)
        free(arrMulDivExpr1)
}
104     arrMulDivExpr => epsilon
{
        Bottom-Up:
        arrMulDivExpr.node_syn = arrMulDivExpr.node_inh
        free(epsilon)
}
105     arrFactor => ID
{
        Bottom-Up:
        arrFactor.node_syn = make_node(ID)
        free(ID)
}
106     arrFactor => NUM
{
        Bottom-Up:
        arrFactor.node_syn = make_node(NUM)
        free(NUM)
}
107     arrFactor => boolConst
{
        Bottom-Up:
        arrFactor.node_syn = boolConst.node_syn
        free(boolConst)
}
108     arrFactor => BO arrExpr BC
{
        Bottom-Up:
        arrFactor.node_syn = arrExpr.node_syn
        free(BO)
        free(arrExpr)
        free(BC)
}
109     addSubOp => PLUS
{
        Bottom-Up:
        addSubOp.node_syn = make_node(PLUS)
}
110     addSubOp => MINUS
{
        Bottom-Up:
        addSubOp.node_syn = make_node(MINUS)
}
111     mulDivOp => MUL
{
        Bottom-Up:
        mulDivOp.node_syn = make_node(MUL)
}
112     mulDivOp => DIV
{
        Bottom-Up:
        mulDivOp.node_syn = make_node(DIV)
}
113     logicalOp => AND
{
        Bottom-Up:
        logicalOp.node_syn = make_node(AND)
}
114     logicalOp => OR
{
        Bottom-Up:
        logicalOp.node_syn = make_node(OR)
}
115     relationalOp => LT
{
        Bottom-Up:
        relationalOp.node_syn = make_node(LT)
}
116     relationalOp => LE
{
        Bottom-Up:
        relationalOp.node_syn = make_node(LE)
}
117     relationalOp => GT
{
        relationalOp.node_syn = make_node(GT)
}
118     relationalOp => GE
{
        Bottom-Up:
        relationalOp.node_syn = make_node(GE)
}
119     relationalOp => EQ
{
        Bottom-Up:
        relationalOp.node_syn = make_node(EQ)
}
120     relationalOp => NE
{
        Bottom-Up:
        relationalOp.node_syn = make_node(NE)
}
121     declareStmt => DECLARE idList COLON dataType SEMICOL
{
        Bottom-Up:
```

```
        declareStmt.node_syn = make_node("Declare", idList.list_head_syn, dataType.node_syn)
        free(DECLARE)
        free(idList)
        free(COLON)
        free(dataType)
        free(SEMICOL)
}
122     conditionalStmt => SWITCH BO ID BC START caseStmts defaultCase END
{
        Bottom-Up:
        conditionalStmt.node_syn = make_node("Switch", make_node(ID), caseStmts.list_head_syn,
defaultCase.node_syn)
        free(SWITCH)
        free(BO)
        free(ID)
        free(BC)
        free(START)
        free(caseStmts)
        free(defaultCase)
        free(END)
}
123     caseStmts => CASE caseValue COLON statements BREAK SEMICOL caseStmts2
{
        Bottom-Up:
        caseStmts.list_head_syn = insert_at_head(caseStmts2.list_head_syn, make_node("Case",
caseValue.node_syn, make_node("Statements", statements.list_head_syn))
        free(CASE)
        free(caseValue)
        free(COLON)
        free(statements)
        free(BREAK)
        free(SEMICOL)
        free(caseStmts2)
}
124     caseStmts2 => CASE caseValue COLON statements BREAK SEMICOL caseStmts21
{
        Bottom-Up:
        caseStmts2.list_head_syn = insert_at_head(caseStmts21.list_head_syn,
make_node("Case",caseValue.node_syn, make_node("Statements", statements.list_head_syn))
        free(CASE)
        free(caseValue)
        free(COLON)
        free(statements)
        free(BREAK)
        free(SEMICOL)
        free(caseStmts21)
}
125     caseStmts2 => epsilon
{
        Bottom-Up:
```

```
        caseStmts2.list_head_syn = NULL
        free(epsilon)
}
126     caseValue => NUM
{
        Bottom-Up:
        caseValue.node_syn = make_node(NUM)
        free(NUM)
}
127     caseValue => boolConst
{
        Bottom-Up:
        caseValue.node_syn = boolConst.node_syn
        free(boolConst)
}
128     defaultCase => DEFAULT COLON statements BREAK SEMICOL
{
        Bottom-Up:
        defaultCase.node_syn = make_node("DefaultCase", make_node("Statements",
statements.list_head_syn))
        free(DEFAULT)
        free(COLON)
        free(statements)
        free(BREAK)
        free(SEMICOL)
}
129     defaultCase => epsilon
{
        Bottom-Up:
        defaultCase.node_syn = NULL
        free(epsilon)
}
130     iterativeStmt => FOR BO ID IN forLoopRange BC START statements END
{
        Bottom-Up:
        iterativeStmt.node_syn =  make_node("For", make_node(ID), forLoopRange.node_syn,
make_node("Statements", statements.list_head_syn))
        free(FOR)
        free(BO)
        free(ID)
        free(IN)
        free(forLoopRange)
        free(BC)
        free(START)
        free(statements)
        free(END)
}
131     iterativeStmt => WHILE BO arithmeticOrLogicalExpr BC START statements END
{
        Bottom-Up:
```

```
	iterativeStmt.node_syn = make_node("While",arithmeticOrLogicalExpr.node_syn,
make_node("Statements", statements.list_head_syn))
	free(WHILE)
	free(BO)
	free(arithmeticOrLogicalExpr)
	free(BC)
	free(START)
	free(statements)
	free(END)
}
132	forLoopRange => forLoopIndex1 RANGEOP forLoopIndex2
{
	Bottom-Up:
	forLoopRange.node_syn = make_node("ForRange", forLoopIndex1.node_syn,
forLoopIndex2.node_syn)
	free(forLoopIndex1)
	free(RANGEOP)
	free(forLoopIndex2)
}
133	forLoopIndex => sign NUM
{
	Bottom-Up:
	forLoopIndex.node_syn = make_node("SignedForIndex", sign.node_syn, make_node(NUM))
	free(sign)
	free(NUM)
}
```