Done: [ 26, 25], [17, 18, 20, 21,22, 23], 24, [10, 11, 12, 13], [14, 9, 8], [19, 7], 6, 4, [16, 15, 5, 3], 2, 1

TBD:

SymbolTable symbab = new SymbolTable()

1. Program ∷ = List<ParamDec> Block
2. ParamDec ∷ = type ident symtab.insert(ident.getText(), ParamDec);
3. Block ∷ = symtab.enterScope() List<Dec> List<Statement> symtab.leaveScope()
4. Dec ∷ = type ident symtab.insert(ident.getText(), Dec);
5. Statement ∷ = SleepStatement | WhileStatement | IfStatement | Chain | AssignmentStatement
6. SleepStatement ∷ = Expression condition: Expression.type==INTEGER
7. AssignmentStatement ∷ = IdentLValue Expression
   1. condition: IdentLValue.type== Expression.type
8. Chain ∷ = ChainElem | BinaryChain
9. ChainElem ::= IdentChain | FilterOpChain | FrameOpChain | ImageOpChain
10. IdentChain ∷ = ident
    1. condition:  ident has been declared and is visible in the current scope
    2. IdentChain.type <- ident.type
    3. ident.type <- symtab.lookup(ident.getText()).getType()
11. FilterOpChain ∷ = filterOp Tuple
    1. condition: Tuple.length == 0
    2. FilterOpChain.type <- IMAGE
12. FrameOpChain ∷ = frameOp Tuple
    1. FrameOpChain.kind <- frameOp.kind
    2. if (FrameOP.isKind(KW\_SHOW, KW\_HIDE) {

condition: Tuple.length == 0

FrameOpChain.type <- NONE

}

else if (FrameOp.isKind(KW\_XLOC, KW\_YLOC){

condition: Tuple.length == 0

FrameOpChain.type <- INTEGER

} else if(FrameOp.isKind(KW\_MOVE){

condition: Tuple.length == 2

FrameOpChain.type <- NONE

} else

there is a bug in your parser

1. ImageOpChain ∷ = imageOp Tuple
   1. ImageOpChain.kind <- imageOp.kind
   2. if (imageOp.isKind(OP\_WIDTH, OP\_HEIGHT){

condition: Tuple.length == 0

ImageOpChain.type <- INTEGER

}

else if (imageOP.isKind(KW\_SCALE)){

condition: Tuple.length==1

ImageOpChain.type <- IMAGE

}

1. BinaryChain ∷ = Chain (arrow | bararrow) ChainElem
   1. Legal combinations shown: in pdf
2. WhileStatement ∷ = Expression Block
   1. condition: Expression.type = Boolean
3. IfStatement ∷ = Expression Block
   1. condition: Expression.type = Boolean
4. Expression ∷ = IdentExpression | IntLitExpression | BooleanLitExpression| ConstantExpression | BinaryExpression
5. IdentExpression ∷ = ident
   1. condition: ident has been declared and is visible in the current scope (lookup() returns non-null)
   2. IdentExpression.type <- ident.type  Type.getTypeName(dec.firstToken)
   3. IdentExpression.dec <- Dec of ident
6. IdentLValue ∷ = ident
   1. condition: ident has been declared and is visible in the current scope
   2. IdentLValue.dec <- Dec of ident
7. IntLitExpression ∷ = intLit
   1. IntLitExpression.type <- INTEGER
8. BooleanLitExpression ∷ = booleanLiteral
   1. BooleanLitExpression.type <- BOOLEAN
9. ConstantExpression ∷ = screenWidth | screenHeight
   1. ConstantExpression.type <- INTEGER
10. BinaryExpression ∷ = Expression op Expression
    1. Legal combinations shown: in pdf
11. Tuple ∷ = List<Expression>
    1. condition: for all expression in List<Expression>: Expression.type = INTEGER
12. **op ∷ = relOp | weakOp | strongOp**
13. **type ∷ = integer | image | frame | file | boolean | url**