1. test1, test5, test16. These test cases have very few nested code blocks and no function calls. This means there are few scopes to track and fewer points of failure

2. test7, test13, test19, test21. These cases have lots of nesting and function calls which means a well implemented stack of symbol tables is critical to passing the test case.

3.

enterProgram(LittleParser.ProgramContext ctx)

enterPgm\_body(LittleParser.Pgm\_bodyContext ctx)

exitPgm\_body(LittleParser.Pgm\_bodyContext ctx)

enterDecl(LittleParser.DeclContext ctx)

enterStr(LittleParser.StrContext ctx)

enterVar\_decl(LittleParser.Var\_declContext ctx)

enterVar\_type(LittleParser.Var\_typeContext ctx)

enterAny\_type(LittleParser.Any\_typeContext ctx)

enterId\_list(LittleParser.Id\_listContext ctx)

exitId\_list(LittleParser.Id\_listContext ctx)

enterId\_tail(LittleParser.Id\_tailContext ctx)

enterParam\_decl\_list(LittleParser.Param\_decl\_listContext ctx)

exitParam\_decl\_list(LittleParser.Param\_decl\_listContext ctx)

enterParam\_decl(LittleParser.Param\_declContext ctx)

enterParam\_decl\_tail(LittleParser.Param\_decl\_tailContext ctx)

enterFunc\_declarations(LittleParser.Func\_declarationsContext ctx)

exitFunc\_declarations(LittleParser.Func\_declarationsContext ctx)

enterFunc\_decl(LittleParser.Func\_declContext ctx)

enterFunc\_body(LittleParser.Func\_bodyContext ctx)

exitFunc\_body(LittleParser.Func\_bodyContext ctx)

enterStmt\_list(LittleParser.Stmt\_listContext ctx)

exitStmt\_list(LittleParser.Stmt\_listContext ctx)

enterStmt(LittleParser.StmtContext ctx)

enterBase\_stmt(LittleParser.Base\_stmtContext ctx)

enterAssign\_stmt(LittleParser.Assign\_stmtContext ctx)

enterAssign\_expr(LittleParser.Assign\_exprContext ctx)

enterRead\_stmt(LittleParser.Read\_stmtContext ctx)

enterWrite\_stmt(LittleParser.Write\_stmtContext ctx)

enterReturn\_stmt(LittleParser.Return\_stmtContext ctx)

enterExpr(LittleParser.ExprContext ctx)

enterExpr\_prefix(LittleParser.Expr\_prefixContext ctx)

enterFactor(LittleParser.FactorContext ctx)

enterFactor\_prefix(LittleParser.Factor\_prefixContext ctx)

enterPostfix\_expr(LittleParser.Postfix\_exprContext ctx)

enterCall\_expr(LittleParser.Call\_exprContext ctx)

enterExpr\_list(LittleParser.Expr\_listContext ctx)

exitExpr\_list(LittleParser.Expr\_listContext ctx)

enterExpr\_list\_tail(LittleParser.Expr\_list\_tailContext ctx)

enterPrimary(LittleParser.PrimaryContext ctx)

enterAddop(LittleParser.AddopContext ctx)

enterMulop(LittleParser.MulopContext ctx)

enterIf\_stmt(LittleParser.If\_stmtContext ctx)

exitIf\_stmt(LittleParser.If\_stmtContext ctx)

enterElse\_part(LittleParser.Else\_partContext ctx)

enterCond(LittleParser.CondContext ctx)

enterCompop(LittleParser.CompopContext ctx)

enterWhile\_stmt(LittleParser.While\_stmtContext ctx)

exitWhile\_stmt(LittleParser.While\_stmtContext ctx)

4. We believe that linked lists should be used for implementing the symbol tables. We would implement a list of lists to store symbol tables.

The main list would store all the symbol tables, and each symbol table and its related variable declarations would be stored in the inner lists

The main list would act as a stack, pushing on new symbol tables as new scopes are opened and popping them off once said scopes are closed.

We think this would be one of the simplest ways to implement the symbol tables, compared to hash tables or trees, making it our main choice.

Using a list stack would make pushing and popping off symbol tables as scopes are entered and exited very simple.

5. A Visitor is more of a manual mechanism that a Listener. Visitor methods must be called explicitly for each child in the parse tree, and failing to invoke a method to visit any child means the subtree will not be visited. A Listener uses a walker object that calls listener methods on its own as previously defined. A Listener has less control of the traversal of the tree as a result.