File - D:\compilers\compilers-antlr\src\main\java\symtable\SymbolTableListener.java

```java
package symtable;

import cymbol.CymbolBaseListener;
import cymbol.CymbolParser;

public class SymbolTableListener extends CymbolBaseListener {
  private final SymbolTableTreeGraph graph = new SymbolTableTreeGraph
();
  private GlobalScope globalScope = null;
  private Scope currentScope = null;
  private int localScopeCounter = 0;

  @Override
  public void enterProg(CymbolParser.ProgContext ctx) {
    globalScope = new GlobalScope(null);
    currentScope = globalScope;
  }

  @Override
  public void exitProg(CymbolParser.ProgContext ctx) {
    graph.addNode(SymbolTableTreeGraph.toDot(currentScope));
  }

  @Override
  public void exitVarDecl(CymbolParser.VarDeclContext ctx) {
    String typeName = ctx.type().getText();
    Type type = (Type) globalScope.resolve(typeName);

    String varName = ctx.ID().getText();
    VariableSymbol varSymbol = new VariableSymbol(varName, type);
    currentScope.define(varSymbol);
  }

  @Override
  public void exitId(CymbolParser.IdContext ctx) {
    String varName = ctx.ID().getText();
    currentScope.resolve(varName);
  }

  @Override
  public void enterFunctionDecl(CymbolParser.FunctionDeclContext ctx
) {
    String retType = ctx.type().getText();
    globalScope.resolve(retType);

    String funcName = ctx.ID().getText();

    FunctionSymbol func = new FunctionSymbol(funcName, currentScope);
    graph.addEdge(funcName, currentScope.getName());

    currentScope.define(func);
    currentScope = func;
  }
```

```
52
53   @Override
54   public void exitFunctionDecl(CymbolParser.FunctionDeclContext ctx
   ) {
55     graph.addNode(SymbolTableTreeGraph.toDot(currentScope));
56     currentScope = currentScope.getEnclosingScope();
57   }
58
59   @Override
60   public void exitFormalParameter(CymbolParser.FormalParameterContext
    ctx) {
61     String typeName = ctx.type().getText();
62     Type type = (Type) globalScope.resolve(typeName);
63
64     String varName = ctx.ID().getText();
65
66     VariableSymbol varSymbol = new VariableSymbol(varName, type);
67     currentScope.define(varSymbol);
68   }
69
70   @Override
71   public void enterBlock(CymbolParser.BlockContext ctx) {
72     LocalScope localScope = new LocalScope(currentScope);
73
74     String localScopeName = localScope.getName() + localScopeCounter;
75     localScope.setName(localScopeName);
76     localScopeCounter++;
77
78     graph.addEdge(localScopeName, currentScope.getName());
79
80     currentScope = localScope;
81   }
82
83   @Override
84   public void exitBlock(CymbolParser.BlockContext ctx) {
85     graph.addNode(SymbolTableTreeGraph.toDot(currentScope));
86     currentScope = currentScope.getEnclosingScope();
87   }
88
89   public SymbolTableTreeGraph getGraph() {
90     return graph;
91   }
92 }
```

```java
package symtable;

import org.antlr.v4.runtime.misc.MultiMap;
import org.antlr.v4.runtime.misc.OrderedHashSet;

import java.util.Set;
import java.util.stream.Collectors;

public class SymbolTableTreeGraph {
  private final Set<String> nodes = new OrderedHashSet<>();
  private final MultiMap<String, String> edges = new MultiMap<>();

  public static String toDot(Scope scope) {
    String symbols = scope.getSymbols().values()
        .stream()
        .map(Symbol::getName)
        .collect(Collectors.joining("</TD><TD>", "<TR><TD>", "</TD></TR>"));

    return scope.getName() +
        " [label = <<TABLE BORDER=\"0\" CELLBORDER=\"1\" CELLSPACING=\"0\">" +
        "<TR><TD COLSPAN = \"" + scope.getSymbols().size() + "\">" + scope.getName() + "</TD></TR>" +
        symbols +
        "</TABLE>>]";
  }

  public void addNode(String node) {
    nodes.add(node);
  }

  public void addEdge(String source, String target) {
    edges.map(source, target);
  }

  public String toDot() {
    StringBuilder buf = new StringBuilder();

    buf.append("digraph G {\n")
        .append("  rankdir = BT\n")
        .append("  ranksep = 0.25\n")
        .append("  edge [arrowsize = 0.5]\n")
        .append("  node [shape = none]\n\n");

    buf.append(String.join(";\n", nodes)).append(";\n\n");

    buf.append(edges.getPairs().stream()
        .map(edge -> String.format("%s -> %s", edge.a, edge.b))
        .collect(Collectors.joining(";\n", "", ";\n"))).append("}");

    return buf.toString();
  }
```

```
51 }
```

```java
package symtable;

import org.antlr.v4.runtime.CharStream;
import org.antlr.v4.runtime.CharStreams;
import org.antlr.v4.runtime.CommonTokenStream;
import org.antlr.v4.runtime.tree.ParseTree;
import org.antlr.v4.runtime.tree.ParseTreeWalker;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.Files;
import java.nio.file.Path;

import cymbol.CymbolLexer;
import cymbol.CymbolParser;

public class SymbolTableListenerTest {
  InputStream is = System.in;

  @BeforeMethod
  public void setUp() throws IOException {
    is = new FileInputStream(Path.of("src/test/antlr/symtable/nested-func.txt").toFile());
  }

  @AfterMethod
  public void tearDown() {
  }

  @Test
  public void testGetAllTokens() throws IOException {
    CharStream input = CharStreams.fromStream(is);
    CymbolLexer lexer = new CymbolLexer(input);
    CommonTokenStream tokens = new CommonTokenStream(lexer);

    CymbolParser parser = new CymbolParser(tokens);
    ParseTree tree = parser.prog();

    ParseTreeWalker walker = new ParseTreeWalker();
    SymbolTableListener symtableListener = new SymbolTableListener();
    walker.walk(symtableListener, tree);

    Path fileName = Path.of("src/test/antlr/symtable/nested.dot");
    Files.writeString(fileName, symtableListener.getGraph().toDot());
  }
}
```