Compilers-lab3 note

实验内容

类型检查

- 本次实验保证了源文件没有语法错误和词法错误，需要检查`SysY`源文件是否存在类型错误，包括：
    1. 变量未声明
    2. 函数未定义
    3. 变量重复声明
    4. 函数重复定义
    5. 赋值号两侧类型不匹配
    6. 运算符类型不匹配
    7. 返回值类型不匹配
    8. ...

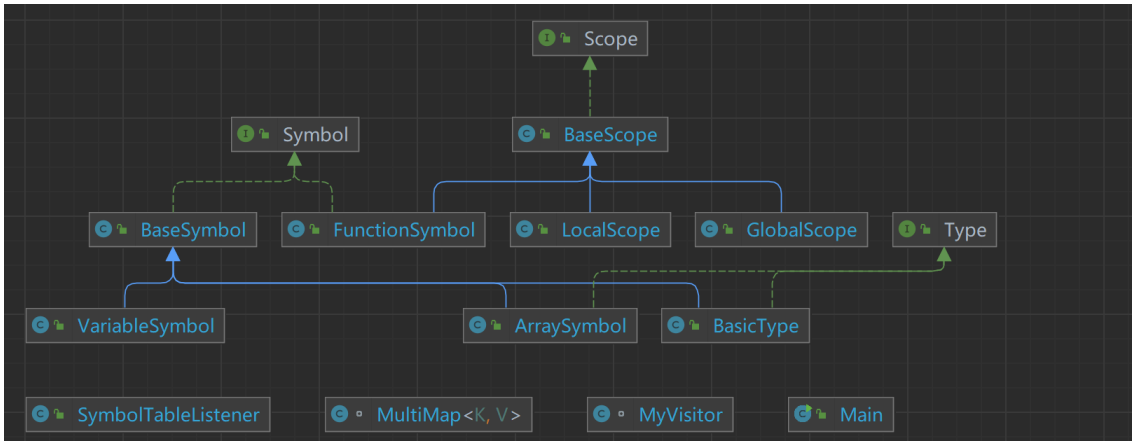
重命名

- 如果经过类型检查后没有错误，就根据输入的行号`lineNo`，列号`column`，变更后的名字`name`，选中文件中的一个变量，对其进行重命名。需要注意的是，必须且只能重命名所有与选中变量为同一个变量的名字，任何被局部作用域覆盖的同名变量不应该重命名。
- 例如下图，提供的输入为`2 8 d`。

```
int main(){
    int a = 0;//rename a into d
    int b = a;//rename a into d
    {
        int a = 1;//not target
        int c = a;//not target
    }
    return 0;
}
```

代码设计

- 类结构如下



- 这次实验在设计类结构时反复调整，起先完全模仿老师课上的演示代码构建类结构，在实际写代码过程中，发现数组类型需要重新构建，并且将所有的`IDENT`分成三类，即`FunctionSymbol`，`VariableSymbol`，`ArraySymbol`，这三个类都继承了`Symbol`，因此它们都有`name`属性来存取`IDENT.getText()`。
- 为了实现对数组类型的解析，没有像实验指导那样递归定义一个数组类型，而是简单的根据`[]`的数量计算数组的维数`dim`，在使用到数组变量时再根据`dim`减去表达式中`[]`的个数得出该数组变量在这里使用的实际维数。代码如下图。

  变量定义

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

  for (int i = 0; ctx.varDef(i) != null; i++) {
    String varName = ctx.varDef(i).IDENT().getText();
    Symbol varSymbol = null;

    int dim = 0;
    for (int j = 0; j < ctx.varDef(i).getText().length() &&
           (ctx.varDef(i).initVal() == null || !ctx.varDef(i).getText().substring(j).equals(ctx.varDef(i).initVal().getText())); j++) {
      if(ctx.varDef(i).getText().charAt(j) == '['){
        dim++;
      }
    }
    if(dim == 0){
      varSymbol = new VariableSymbol(varName, type);
    } else {
      varSymbol = new ArraySymbol(varName, type, dim);
    }
```

  变量使用（这段解析数组维数的代码在多个涉及到`IDENT`使用的地方都用到了，但由于每个`ctx`的结构有所差异，无法抽象为一个函数）

```
int dim = 0;
for (int i = 0; ctx.L_BRACKT(i) != null; i++) {
  dim --;
}
if(currentScope.resolve(ctx.IDENT().getText()) instanceof ArraySymbol){
  dim += ((ArraySymbol) currentScope.resolve(ctx.IDENT().getText())).getDim();
}
if(dim < 0){
  //Error 9
  Main.ErrorExist = true;
  System.err.println("Error type 9 at Line " + ctx.IDENT().getSymbol().getLine() + ": Not an array: " +  ctx.IDENT().getText() + ".");
  return;
}
```

- 为了存储每个变量每次出现的行号和列号，手动实现了一个`MultiMap`。

遇到的问题

- 在错误检查时，曾经因为在多个地方检查同一个`Error Type`，导致同一个错误输出多次，经过本地反复测试才发现问题。
- 在错误检查时未找到合适的方法在Listener中通知Visitor是否存在类型错误，最后只能通过Main中的全局变量实现该功能。

在`SymbolTableListener.java`中

```
@Override
public void enterLVal(SysYParser.LValContext ctx) {
  if(currentScope.resolve(ctx.IDENT().getText()) == null){
    //Error 1
    Main.ErrorExist = true;
    System.err.println("Error type 1 at Line " + ctx.IDENT().getSymbol().getLine() + ": Undefined variable: " +  ctx.IDENT().getText() + ".");
    return;
  }
```

在`Main.java`中:

```
  if(!ErrorExist) {
    MyVisitor visitor = new MyVisitor();
    visitor.visit(tree);
  }
```