lab5: Function and Var

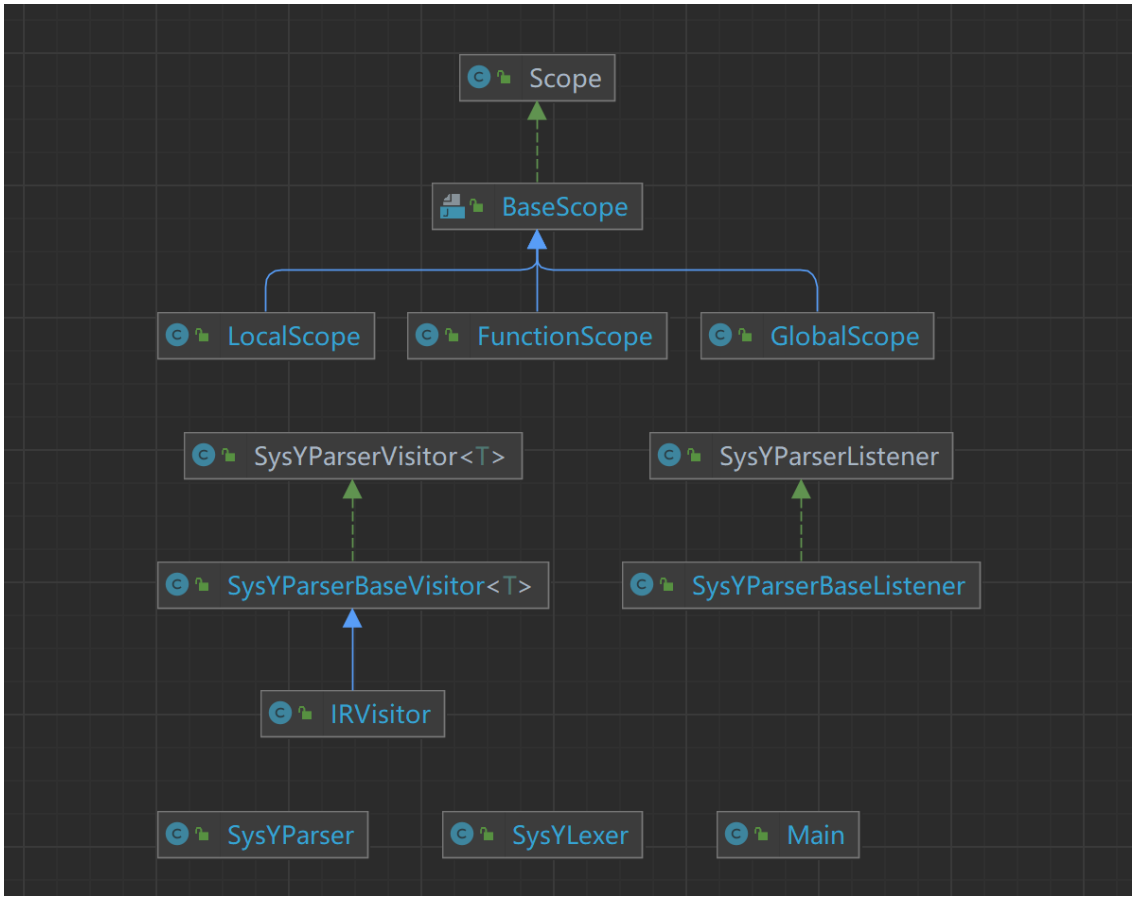
赵耀 201870139

实验内容

- 本次实验借助LLVM java api来翻译SysY语言的函数与局部变量。
- 对于函数的翻译，涉及到函数的声明与调用，需要分别重写`visitFuncDef`和`visitCallFuncExp`两个方法，并且除了生产函数定义和调用的IR，还要将函数的ValueRef添加到符号表中。
- 对于局部变量的翻译，涉及局部变量的声明，定义，二次赋值，访问，需要分别重写`visitVarDecl`、`visitAssignStmt`、`visitLvalExp`。需要注意的是，常量的声明、定义、赋值均与变量相似，不再赘述。

代码设计

- 代码结构



- 本次实验对于lab3的符号表重新设计，将原本的String-Symbol映射修改为String-LLVMValueRef映射，从而与LLVM的API兼容。并且由于本次实验只涉及一维数组，没有必要创建一个ArrayType类，因此我选择通过一个String-Integer的映射来获取每一个变量（常量）的dim，即维数，数组的维数为1，其他均为0。

```
public class BaseScope implements Scope{
    private final Scope enclosingScope;
    private final Map<String, LLVMValueRef> symbols = new LinkedHashMap<>();
    private String name;
    private final Map<String, Integer> dimMap = new HashMap<>();
```

- 本次实验对于IR的生成，最难的部分是对于数组的初始化和读取，在代码中通过三个方法`buildGEP`、`loadGEP`、`storeGEP`分别负责初始化数组、根据索引获取元素、根据索引给元素赋值。
- 虽然本次实验未涉及在函数的内部return，但为了代码的可用性，我仍然进行了相应的处理，可以实现从函数的某个`LocalScope`，不断递归向上，直到作用域指向的是`FunctionScope`，如下图所示。

```
@Override
public LLVMValueRef visitReturnStmt(SysYParser.ReturnStmtContext ctx) {
    Scope temp = currentScope;
    while(!(temp instanceof FunctionScope)){
        temp = temp.getEnclosingScope();
    }
    if(functionRetTypeMap.get(temp.getName()) != null){
        if(functionRetTypeMap.get(temp.getName()).equals("int")) {
            LLVMBuildRet(builder, visit(ctx.exp()));
        } else {
            LLVMBuildRetVoid(builder);
        }
    }

    return null;
}
```

遇到的困难

- 首先是对LLVM API的不熟悉，通过阅读文档、学习助教代码、在github上查找API使用的源码学习。
- 其次是对于GEP指令的疑惑，在不断尝试和debug后，了解了GEP指令的各个参数的含义。

```
public static org.bytedeco.llvm.LLVM.LLVMValueRef LLVMBuildGEP(
    LLVMBuilderRef llvmBuilderRef,//builder
    LLVMValueRef llvmValueRef,//变量Ref
    PointerPointer pointerPointer,//指向变量的“指针”结构
    int i,//指针结构的大小
    String s //变量名
)
```

- 最后还有从listener转到visitor来构造符号表的不适应，不清楚如何在visitor中实现enter和exit的功能，在查阅资料后了解到，在visitor中可以借助`visit()`方法来访问任意节点就可以先访问子节点然后再退出。