编译原理 lab5

一、项目结构：

本次实验在上回 lab4 基础上进行，除了 lab4 已存在的两个.g4 文件，main 文件以及 LLVMVisitor.java 外，由于本次实验需要符号表，新增了 Scope 接口，BaseScope 实现 Scope，和继承自 BaseScope 的 LocalScope 和 GlobalScope。

二、实验思路

本次实验需要解析变量和函数，故需要使用符号表。符号表具体实现思路继承自 lab3，但由于 LLVM 自带相应的 type，故删去了自定义的 Type 和 Symbol 类。Scope 的 map 中存放的直接是 LLVMValueRef，较为方便。

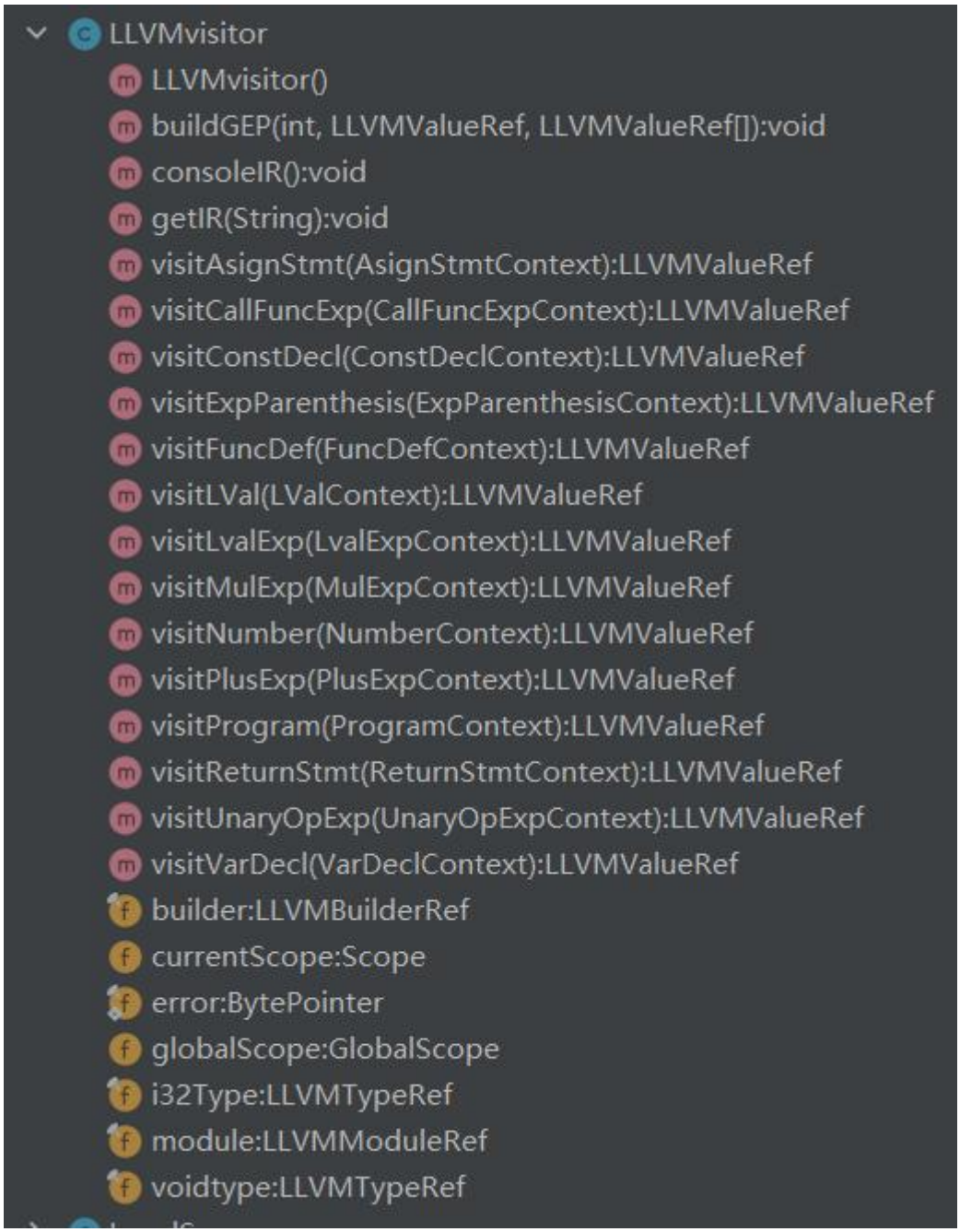
之后则开始在 LLVMvisitor 类中重载相应的关于函数和变量的方法，数组需要使用到 GEP 较为复杂，常量定义也不能少。

三、具体实现如下：

Scope 主要方法如下：

```
interface Scope {
    public void setName(String name);
    public Map<String, LLVMValueRef> getLLVM();
    public Scope getEnclosingScope();
    public void define(String name, LLVMValueRef llvmValueRef);
    public String getName();
    public LLVMValueRef resolve(String name);
}
```

LLVMvisitor 主要函数如下：



构造函数以及操作符和上次实验基本相同，本次实验主要是 visitFuncDef，visitCallFuncExp, visitVarDecl，visitLvalExp，visitAsignStmt , visitConstDecl 这几个函数

1. 首先 visitFuncDef，由于本次实验函数返回值只有 int 或 void(对应 LLVM 的 i32type 和 voidtype), 参数都只有 int 故此方法实现较为简单，思路与 lab3 别无二致，具体实现见代码。只是在跑 oj 时发现返回值为 void 的函数，如果没有 return 语句，也要有对 return 的解析，但 visitReturnStmt 在没有 return 语句时并不能起到作用，故在 visitFuncDef 函数末尾加上对返回值的判断，若为 void 则加上一个 LLVMBuildRetVoid(builder)，否则返回值是 int，由 visitReturnStmt 处理

2. 在调用函数时的访问到的 LLVMBuildCall 中，主要使用 LLVMBuildCall 的 api。首先在当前作用域解析函数名，得到函数，之后建立 PointerPointer<Pointer> args 并将实参放入其中，顺便获取实参数量，最后将上述三个以及 builder 都传入 LLVMBuildCall 中并返回即可。

Ps:但在跑 oj 时发现若函数返回值为 void，则 LLVMBuildCall 的最后一个参数，字符串要为空。

3. visitVarDecl 和 visitConstDecl 基本没有什么差别，只不过一个是常数一个是变量，故只介绍 visitVarDecl。在变量定义时，由于 visitVarDecl 中可能包含多个变量连续定义，即包含多个 varDef，故首先要遍历 varDefContext。
之后直接给变量开辟一个内存空间，赋值变量指针，然后判断每个 varDefContext 是否有赋值语句，若没有则在当前作用域定义并结束循环。若有则继续判断是否是数组，若为变量则直接将 visit 赋值语句(initVal)得到的结果通过 LLVMBuildStore 存入之前变量指针。
若为数组则较为麻烦：首先要获得数组长度，重新创建一个符合数组长度的空间并作为 vectorPointer，之后将 initVal 中{}中的值遍历放入一个长度与解析数组长度相同的一维数组 initArray 中，并将剩下的值置为 0。之后则遍历 vecPointer 将其中的每个元素的指针通过 LLVMBuildGEP(此 api 的使用较为复杂，第一个参数为 builder；第二个为 vectorPointer；第三个为一个非常奇怪的 PointerPointer<LLVMValueRef>，它有两个元素，第一个是 LLVMConstInt0，第二个是 LLVMConstInti（i 为想要得到的指针的索引）；之后则是一个固定的 int 值 2 和操作名字符串)得到，之后遍历 initArray，通过 LLVMBuildStore 将 initArray 中的每个元素的值按索引存入每个指针所指的空间中。
Ps：上述是主要实现思路，具体实现上为了方便，将 Gep 的循环调用写到了函数 buildGep 中
4. 在 visitLval 中，通过解析变量名得到指针，返回的分别是变量或数组的某个元素的**指针，**之后在 visitLvalExp 中对指针进行 LLVMBuildLoad 操作获得实际值。
5. visitAsignStmt 则只使用简单的 LLVMBuildStore 将右值存入左值的指针中。