编译原理lab5

1. 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。



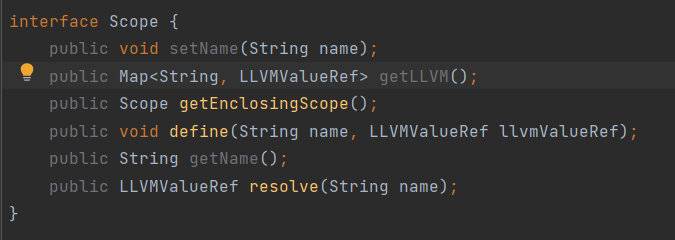
1. 实验思路

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。

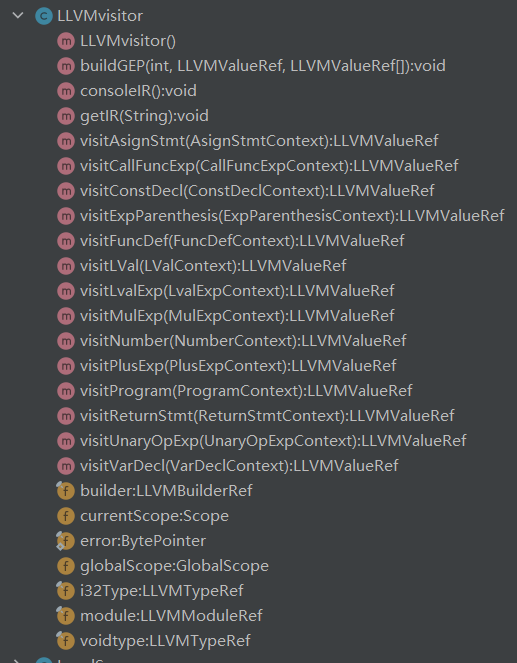
之后则开始在LLVMvisitor类中重载相应的关于函数和变量的方法，数组需要使用到GEP较为复杂，常量定义也不能少。

1. 具体实现如下：

Scope主要方法如下：



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。

1. 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

1. 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。
2. visitAsignStmt则只使用简单的LLVMBuildStore将右值存入左值的指针中。