编译原理 Lab5 实验报告

1 程序功能

实验五在实验四的基础上，目标仍然是实现有语义规则的函数解析。实现了函数的定义与调用，局部变量的定义和调用，最后可以返回 main 函数中 return 语句的正确值。

2 程序实现

代码结构：
一个检查错误的方法类 checkVisitor，用于对于语法树的深度遍历访问
基本变量的类型类 Type：包括 int 类型，array 类型，function 类型，存放 LLVMValueRef 指针的 pointer
创建符号类 Symbol：BaseSymbol 为基础符号类型，functionSymbol 类继承自 BaseScope 同时实现 Symbol 类
符号表类 Scope：基础符号表类 BaseScope，globalScope、functionSymbol 和 localScope 继承自基础符号表类

三个类的关系：Type 中包括了变量类型种类名，Symbol 类中包括了变量名和变量类型，以及是否需要重命名，Scope 为作用域类型，其中包括了作用域名，作用域的返回值以及父作用域，其中，每个变量的值或者返回值都存放在 type 类中

代码实现过程：

```
String leftVarName = ctx.IDENT().getText();
        Type leftVarType; // 左值
        int number;

        List<TerminalNode> dimension = ctx.R_BRACKT();
        if (dimension.size() == 0) { // 左侧为变量
            leftVarType = new Type("int");
            leftVarType.pointer = LLVMBuildAlloca(builder, i32Type, leftVarName);

            // 对变量进行赋值
            if (ctx.ASSIGN() != null) {
                LLVMValueRef rightVal = getNumber(ctx.initVal().exp());
                int x = (int) LLVMConstIntGetSExtValue(rightVal);
                //leftVarType.pointer = rightVal;

                LLVMBuildStore(builder, rightVal, leftVarType.pointer);
                LLVMValueRef value = LLVMBuildLoad(builder, leftVarType.pointer, "value");
                x = (int) LLVMConstIntGetSExtValue(value);
                x = 1;
            }
```

```
21.         }
```

以变量定义并初始化为例，每次定义变量时，先将 pointer 申请一块内存区域，然后通过 getNumber 函数得到等号右侧的值，再通过 store 将当前得到的右侧值存储到指针当中
这样便可以在定义每一个变量的时候能够保存申请的到内存区域并且保存当前内存区域里面的值。

对于变量，需要在变量定义的时候给变脸中的 pointer 指针进行内存申请。而变量的赋值可能在初始化阶段，也可能在赋值语句阶段

```
22. result = LLVMBuildCall(builder, funcScope.type.pointer, actualParam, actualParam.size(), "")
    ;
```

对于函数调用，需要使用 LLVMBuildCall 函数得到具体实参的返回值

3 精巧设计

实验五代码是对于实验四的代码进行扩充和改善，结合了之前语义规则的数据结构，在语义规则实验框架部分在每一个变量的 type 内添加 pointer 指针来保存每一个 LLVM IR 的值。

4 印象深的地方

对于函数调用部分，之前我的实现思路是在调用函数的地方再次遍历被调用函数节点，但是这样会出现段错误，后来使用 LLVMBuildCall 函数可以直接得到函数内的返回值。