# 编译原理实验：Lab4

指导老师：徐辉

2021 年秋季学期

## 1 实验介绍

在完成了 lab3 之后，我们已经得到了一个较为完整的可使用的编译器，不过这个编译器所支持的语法还过于简单，因此其可以编译的代码的功能有很大的局限性，Lab4 的实验内容就是要帮助这个简单的编译器支持更多的语法，使得我们可以编译更为复杂的程序。

### 1.1 语法产生式

本实验完整的语法产生式如表 1所示：

其中，加粗部分为本次实验的语法产生式与之前不同的地方，主要是对可编译的代码增加了以下几个支持点：

- 在 body 中支持多个 statement，不再仅仅是一个表达式，且每个 statement 都是以分号结尾的。
- <statement> 不再只是 expr，它分为三种类型，也就是新增的三条产生式 <decl>，<simp>，<return>。
- <decl> 为声明语句，例如 $int\ i, j$。
- <simp> 为简单语句，这里可以理解为赋值语句，语句中的操作符只会为 = 赋值操作。
- <return> 为返回语句，跟传统意义上的 return 语句是一样的，将对应的 <expr> 作为返回值。

### 1.2 实验内容

本次实验由于是对编译器增加一系列完整的语法支持，因此要修改的部分是贯穿整个编译器的。

首先对于词法分析部分，必须要识别新的关键字 return，而在语法分析部分，则可能需要新建多个新的 class，比如 $DeclExprAST$,$ReturnAST$ 等，新增 class 的结构可以参照之前的那些 class 来实现，在新增完后还需修改语法分析器的逻辑，从而正确地 parse 这些新的类。之后便是中间代码生成部分，主要就是对这些新的 class 的 codegen 函数实现重载，并修改一些相关的 codegen 函数。

这次实验 codegen 部分中最需要注意的点就是局部变量值的保存。赋值语句以及声明语句需要完成值的存储这项工作，因此要执行 $Builder- > CreateStore()$ 的操作，同时还要注意的是

| | | |
|---|---|---|
| $< program >$ | $::=$ | $< gdecl >^* < function >^*$ |
| $< gdecl >$ | $::=$ | $extern < prototype >;$ |
| $< function >$ | $::=$ | $< prototype >< body >$ |
| $< prototype >$ | $::=$ | $< type >< ident > (< paramlist >)$ |
| $< paramlist >$ | $::=$ | $\epsilon \mid < type >< ident > [, < type >< ident >]^*$ |
| $\boldsymbol{< body >}$ | $\boldsymbol{::=}$ | $\boldsymbol{\{[< stmt >]^*\}}$ |
| $\boldsymbol{< stmt >}$ | $\boldsymbol{::=}$ | $\boldsymbol{< simp >; \mid < return >; \mid < decl >;}$ |
| $\boldsymbol{< decl >}$ | $\boldsymbol{::=}$ | $\boldsymbol{< type >< ident > [, < ident >]^*}$ |
| $\boldsymbol{< simp >}$ | $\boldsymbol{::=}$ | $\boldsymbol{< ident >=< exp >}$ |
| $\boldsymbol{< return >}$ | $\boldsymbol{::=}$ | $\boldsymbol{return < exp >}$ |
| $< exp >$ | $::=$ | $(< exp >) \mid < const > \mid < ident > \mid$ |
| | | $< exp >< binop >< exp > \mid < callee >$ |
| $< callee >$ | $::=$ | $< ident > (\epsilon \mid < exp > [, < exp >]^*)$ |
| $< ident >$ | $::=$ | $[A-Z\_a-z][0-9A-Z\_a-z]^*$ |
| $< const >$ | $::=$ | $< intconst > \mid < doubleconst >$ |
| $< binop >$ | $::=$ | $+ \mid - \mid * \mid <$ |
| $< intconst >$ | $::=$ | $[0-9][0-9]^*$ |
| $< doubleconst >$ | $::=$ | $< intconst > . < intconst >$ |
| $< type >$ | $::=$ | $int \mid double$ |

表 1: 语法产生式

重复同名变量的声明，后声明的要覆盖前声明的，比如声明了两次 temp，第一句是 *inttemp*，第二句是 *doubletemp*，那么 temp 应该在后面的使用是作为 double 类型的。相关的操作内容可以参考 LLVM Tutorial 的相关实验以及查阅官方 API 文档。

## 2 实验测试

lab4 实验的测试用例为压缩包中的 *input_example.data*, *main.cpp* 以及 *output_example.data*, 在配置好环境后，在当前目录的命令行使用 lab3 中相同的编译方式可得到输出结果，可将其与 *output_example.data* 进行对比。

本次实验是要在 lab3 基础上进行修改的，因此没有额外的代码框架。

## 3 实验提交

实验完成之后，将实验代码在截止日期之前上传至 elearning，助教将根据代码完成质量以及测试用例通过情况进行打分。

提交时只需要提交完整可编译运行的代码文件，将代码压缩到以学号命名的压缩包中 (zip 格式)，提交压缩包文件。代码的文件名为 lab4.cpp。