编译原理实验报告二

陈永恒 151220012

前言

这次使用了LLVM的库来完成实验，主要包括:

```
#include <llvm/IR/Value.h>
#include <llvm/IR/Module.h>
#include "llvm/IR/Function.h"
#include "llvm/IR/IRBuilder.h"
#include "llvm/IR/LLVMContext.h"
#include "llvm/IR/Type.h"
#include "llvm/IR/Verifier.h"
```

从LLVM的文档可以发现，LLVM并没有把这一步检查错误单独提出来，而是和代码生成结合在了一起。这使得有些LLVM的功能不能直接满足这次的实验要求，例如LLVM中structure的成员并没有记录变量名，给实验增加了难度(LLVM挺强大的，但是感觉做这个实验不用LLVM更加好做一点orz..)。

实现的功能

1. 所有的必做功能。

实验说明

符号表的实现

LLVM中提供了LLVMContext和Module模块，其中LLVMContext提供了上下文信息，包括可定义的类型和唯一的常量，Module则定义了一个程序的所有东西:符号表，全局变量，函数等。因此我不用额外定义符号表(除了定义一个结构体来记录structure的成员名字)。

定义变量

在LLVMContext和Module都初始化以后，定义一个变量的方法如下:

```
1. TheModule->getOrInsertGlobal(var_name,var_type);
```

这样类型为 `var_type` 的变量 `var_name` 就被添加到了符号表里面。

定义函数

```
1. TheModule->getOrInsertFunction(function_name,function_type);
```

看起来好像都很简单，但是难点在于如何创建function_type，并可以进行语义分析。

语义分析的方法

为了进行语义分析，我给每一个Exp类型都赋予了一个type，这个type可以包含各种属性，例如是否为右值，是否为 `int` 或 `float` ，是否为 `struct` .这样子就可以在更高一层的表达式中做语义分析了。

目录列表

目录列表解释如下

```
1. Project 2  #实验二主目录
2.     main.cpp    //主程序
3.     Makefile    //编译文件
4.     tokens.l    //Flex部分源代码
5.     parser.y    //Bison部分源代码
6.     node.h      //语法树中每一种node的定义
7.     node.cpp    //对node的打印函数的定义,语义检查的函数，还有一些辅助函数，
   如createVar(type,name)
8.
9.     report.pdf //实验报告，即本文件
10.
11.    test_case
12.        *       #各种测试文件
```

编译与运行

编译命令: `make`

运行命令: `./parser file`