# 实验三 中间代码生成

201220176 刘兴

## 一、实现功能

1．完成附录中**最基本的中间代码生成**。

**2．**完成要求 3.1—**修改前面对 C--源代码的假设 2 和 3，使源代码中：**

**a) 可以出现结构体类型的变量（但不会有结构体变量之间直接赋值）。**

**b) 结构体类型的变量可以作为函数的参数（但函数不会返回结构体类型的值）。**

**3．**完成要求 3.2—**修改前面对 C--源代码的假设 2 和 3，使源代码中：**

**c) 一维数组类型的变量可以作为函数参数（但函数不会返回一维数组类型的值）。**

**d) 可以出现高维数组类型的变量（但高维数组类型的变量不会作为函数的参数或返回类值）。。**

## 二、如何编译

1．通过 makefile 进行编译。

## 三、设计思路以及数据结构

### 1．数据结构

（1）修改了 Operand 数据结构如下，方便对不同的操作数实行不同的操作。

```
struct Operand_ {
    enum { VARIABLE_O, TEMP_O, PARAMETER_O,FUNCTION_O,CONSTANT_O,
LABEL_O } kind;
    enum { VAL_O, ADDRESS_O} type;
```

```
    int isfunctionpara;      //仅当该符号出现在符号表中，被判断为变量时使用，
因为当其出现在函数参数时，值为真正的地址；而局部变量则只为名字。
    union {
        int var_no;
        int value;
        char* func_name;
    } u;
};
```

（2）InterCode 数据结构也根据文法进行相应的扩展，主要是对于 assign 区分了不同的类型。

2．设计思路：

（1）最基础的要求：只需要根据文法，为各个非终结符号构造对应函数，然后按照文法的结构进行相应的调用。在整个翻译过程中，需要根据各种翻译方案，在恰当的位置进行翻译并生成代码保存在 InterCode 链表种即可。最后将链表按照格式打印到输出文件中即可。

（2）拓展要求：其实并没有多干些什么，只是在各个翻译方案中，注意每个变量、临时变量的类型到底是地址还是值。其中最容易出错的地方是传参的过程中采用传地址时，要和局部变量进行区分。我在 Operand 种增加 int isfunctionpara; 仅当该符号出现在符号表中，被判断为变量时使用，因为当其出现在函数参数时，值为真正的地址；而局部变量则只为名字。

# 四、遇到的问题

在这第二次编译原理的实验中，还是遇到了诸多的问题。

首先还是刚上手时的不知所措，在看了很长一段时间才逐渐知道

该怎么做。

主要的问题就是空指针的引用问题，在设计时出现了太多的漏洞导致经常会出现空指针的引用而引起段错误，导致实验一直卡壳。第二个问题就是 DEBUG 的过程中真的是毫无头绪，尤其是 E2-3 中出现了不应该出现的数组赋值，花了我大量的时间找到了这个问题。这次 OJ 给的输出实在太少，DEBUG 相当的困难。