编译原理实验三

姓名：殷天润 学号：171240565 邮箱：171240565@smail.nju.edu.cn 班级：基础班

项目环境:

1. GNU LINUX Realease: Ubuntu 18.04.4 LTS,5.3.0-40-generic,Xming
2. GCC version:7.5.0
3. GNU Flex version:2.6.4
4. GNU Bison version: 3.0.4

实现功能

1. 必做内容+选做内容
2. 大体介绍：我才用的是双向链表构造中间代码，因此实验三的本质就是借助抽象语法树+实验二得到的符号表信息，通过一定的规则推导出中间代码；

   中间代码的结构：操作符+类型+辅助信息(比如char * relop)

   操作符的结构：类型+ifaddress(表示是否需要加*或者&)+辅助信息(包括变量名,函数名,number,深度),其中深度是用于数组中间代码生成的关键信息;

   我是用add_inter函数往中间代码双向链表插入结点,使用new_op构造新的操作符,使用new_intercode构造新的中间代码;在new_op中,对于临时变量OP_TEMPVAR,如果ifaddress是OP_ADDRESS那么就在前面加上*;对于变量OP_VARIABLE,如果ifaddress是OP_ADDRESS那么就在前面加上&;通过这样的处理可以减少用于表示地址的重复指令;但是这种操作也给后来的一些数组,函数调用,结构体带来了额外的判断;
3. 实验二文件的一些改动：
   1. 我在实验二的文件semantic.c中新建了read,write函数;
   2. 在实验二结束之后根据发放的超级测试样例debug,解决了超强样例SF的问题;在实验三的过程中还发现之前结构体结构的一个bug,这个bug会导致int a,b,c;这样的域在连接结构体的field的时候缺少b,c这样的元素;
   3. 因为实验三的假设,在insert_struct的时候我不在在结构体的域后面添加结构体元素名再插入结构体表(struct a{int b;}原来插入的是b a,现在是插入b;)并且现在这些域不仅会插入结构体表struct_table还会插入符号表symbol_table方便查询;
   4. 对于symbol node结构体还添加了belongtostructname方便查阅结构体的域属于哪一个结构体,添加了offset,这个属性仅仅对于结构体中的元素有用,用来指示这个元素相对于结构体首部的偏移量,在插入的时候进行更新;我也因此修改了Def_struct函数和specifier函数的部分代码;
4. 实验三的一些设计：
   1. Stmt和cond部分:我根据课本6.6.5 避免冗余的goto指令部分进行了优化,这部分的用意我以下面的例子说明,对于WHILE LP Exp RP Stmt

      讲义版本:

      ```
      label4:
      if cond goto label5(true label)
      goto label6 (false label)
      label5:...
      goto label4
      label 6:...
      ```

```
等价于
label4:
if !cond goto label6(false label)
....(true contant)
goto label4
label 6:
...
```

根本在于使用!cond也就是对条件取反来减少一次goto操作;并且条件中的一些IFGOTO操作我也通过一些判断简化成了GOTO操作,比如条件中的INT部分;

2. 数组结构体处理:

- 我是用gettypesize函数确定一个type的大小;这是一个递归实现的函数,对于结构体变量是一个相加的操作,对于数组变量是一个相乘的操作;
- 数组处理有两个部分,一个是VarDec_g部分,这个部分主要就是通过ID确定数组的type,然后通过gettypesize来确定要申请的空间的大小;在这一步我还会更新查询到的symbol node的ifaddress,更新为OP_VAR用于后续的处理;
- 数组处理的另一个主要部分是Exp_g部分;这个地方的处理我做过改动,一开始我是用循环做的处理,通过wihle以及抽象语法树可以得到底部的ID结点,进而得到ID的type然后一步步计算偏移值,这个方案有一个缺陷在于对于a.b.c[10]这样的变量,循环一路往下得到的是a这个ID而不是C； 因此我**目前**的方案还是**递归**的方案,也因为这个方案我往Operand操作符结构体里面添加了depth变量;主要的操作是在Exp_g中的ID和ID DOT EXP部分添加一个返回的临时变量TEMPVAR的depth和varname的操作,这样在最底层的Exp LB Exp RB就可以获得depth初始化和varname信息;通过varname可以查询符号表得到数组结构symbol node,进而得到单位偏移量(比如int 是4,结构体数组是结构体的大小),然后可以结合depth得到当前的偏移,然后相乘得到最终偏移量,这一层相当于处理完了然后新建一个临时变量用于保存和传递到上一层,depth+=1;
- 我还做了数组相互之间的赋值,具体实现在Exp_g的ASSIGNOP部分,我实现的功能的类似于memcpy,大致的算法举例如下:

```
DEC v0 24
DEC V1 24
......
t99 := &v1 //表示a[0] 可以加偏移量
t100 := &v1 + #24 //表示空间终点;
t101 := &v0 //表示b[0] 可以加偏移量
LABEL label5:
IF t99 >= t100 GOTO label4
*t101 := *t99
t99 := t99 + #4 //指针偏移
t101 := t101 + #4 //指针偏移
GOTO label5
LABEL label4:
```

大概就是用t101表示被赋值的起始点,t99表示用于赋值的起始点,t100表示用于赋值的终点,用一个循环操作完成赋值;

- 在Arg_g中也就是函数调用中我特地增加了一个判断用于$int\ a(int\ t[5])$; $b[5][5]$传递使用$a(b[5])$这种操作,主要的实现是当判断param是ARRAY,通过符号表查询varname得到相应的数组结点,然后计算数组的深度,和当前传递的操作符的depth进行比较,然后判断ifaddress应该怎么赋值;该问题debug的时候找了很久;

3. 结构体的处理:

结构体的操作其实有很多工作实在前面提到的修改lab2部分做的,基于这些工作通过查询对应的ID就可以得到相应的ID关于结构体头部的偏移,然后使用一个TEMPVAR记录以及加上前面的exp的偏移就行了;

5.主要的bug:

1. 各种数组,结构体的ifaddress怎么赋值的问题,这些用一些简单的案例就可以找出来,主要是逻辑上的困难;
2. a.b.c[10]这样的问题,导致了我数组的重构;
3. Arg_g一开始实现的时候因为Exp_g没有实现完善会返回null,就加了一句if(Exp_g(...)!=NULL)结果这句会产生额外的副作用导致Exp_g执行两遍,然后这个bug很隐蔽,特征在于直接用n=prt(n,a)和mod(n=prt(n,a),2)两个代码之后n的值不一样;
4. 功能上的bug,比如数组赋值和传递数组;
5. 主要debug的方式是用printf+write+测试样例局部化;
6. 总结:比调C代码TLE的bug更困难的是什么问题?答:调C代码中间代码TLE的bug.T_T

编译方法

使用make编译,make test生成中间代码out1.ir在当前目录中,如果使用irsim的C版本可以使用./irsim out1.ir执行,使用python版本gui执行;

反馈

1. 没想到debug那么花时间,优化做的不太多有点遗憾;感谢出测试的同学！在使用测试和将测试本地化(提取错误部分)过程中感觉把测试主题化挺重要的;
2. 这次没用gdb调bug,想听一听其他同学相关的经验;