编译原理实验二

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
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. 完成了必做内容+所有的选做内容
2. 主要的实现包括符号表的实现和具体语义分析的实现;
3. 符号表的实现:
    1. 主要实现在symbol_table.c中,我有两个符号表,一个用来存包括: 函数名,结构体名,变量名在内的符号,称为符号表A;一个用来存放结构体的域名,称为符号表B;
    2. 符号表A和符号表B主体都是哈希表,使用讲义给的函数计算名字的哈希值;符号表B中的名字在进入符号表之前会进行额外的处理,即将结构体名添加到域名的后面;比如struct A{int a;};中的a填入符号表B之前会先变成aA然后再加入;这样的设计主要是考虑到域名的要求,检验错误15,方便查询;并且因为这样的组合名字最长到64位,我用的char指针来进行储存;
    3. 符号表A的主体是哈希表,我使用桶也就是一个链表来防止重名或者冲突;为了体现局部作用域,我另外开了一个全局的链表设计成了一个栈,每一次进入局部作用域的时候新建一个栈结点并且返回给上层;每一次插入元素的时候,首先插入到哈希表中(桶使用头插,新加入的元素在头部),然后再根据栈结点找到对应栈的末尾元素,然后把对应的元素加进去,这样的结构是类似于十字链表的;当退出局部作用域的时候,就根据栈尾的元素进行遍历,然后再哈希表里面删除对应的元素,free相应的空间,然后删除对应的栈结点;
    4. 符号表中的每一个node都有kind(区分变量,结构体名,符号名),lnext(哈希表链表),cnext(局部作用域链表),field(存储type,名字),ifdef(区分声明),depth(深度,确定作用域);
    5. 我有两个主要的查询符号表A的函数,一个是query_symbol,一个是query_symbol_exist,一个是返回当前depth的符号,一个是返回<当前depth的桶的第一个符号;这样可以解决全局符号,局部符号的使用问题;
    6. 另外我还在这个文件中实现了比较两种type类型的check_type函数,主要用于比较两种type是否结构等价;其中在和同学的讨论中,我认为array这个类型其实有两种等价标准,一个是弱等价,用于两个array的比较,只需要深度一样即可,比如int $a[99][100], b[3]$中,$b = a[97]$是合法的;还有一个是强等价,用于结构体结构等价比较中的array,此时结构体等价必然要求array的size和深度都要相同才行;
    7. 我额外实现了show_global_table();show_scope_table;一个是横向一个纵向的打印符号表A进行debug;
4. 语义分析部分:semantic.c中
    1. 主体结构: dfs; 我所有的语义分析都是基于lab1的语法树做的,宏观而言的工作是从语法树中获得信息,然后进行填表,查表分析;大部分调用的函数我都采用递归实现,少部分需要进行额外链表链接的部分,比如structspecifier部分我使用了循环实现,通过多个指针+while的结构完成了struct中链表的组装工作;
    2. 我在语义分析文件中维护了一个$depth_$,初始也就是全局的时候是0,每进入一个compst加一,在插入符号表的时候都会携带这个信息来维护变量的作用域;通过$depth_$和上述的用于符号表A的局部链表结构,我完成了局部作用域的实现;

3. 对于每一个变量都有ifdef用来标记是声明还是定义,这个主要用于实现要求2.1,为了实现这个要求,我在插入每一个函数声明的时候，除了基本的类型，参数检查外，我会检查符号表中是否已经有了该函数的声明或者定义，如果有了就不再插入这个声明;因为我的符号表A的query是返回该桶中的第一个符号，这样的设计可以防止声明-定义-声明导致的误认为该函数还是声明的错误;

   另外当插入函数声明的时候，我还在symbol_table.c中开了一个链表用来记录声明过的函数,当程序检查完之后遍历这个链表检查所有声明过的函数是否被定义;

   我觉得要求2.1和错误类型2有一定的冲突，当出现声明a;调用a;定义a;这种结构的时候调用a如果抠字眼的话是犯了错误类型2的错误的,我是根据gcc的通用情况，不报这样的错误;**我觉得要求2.1的讲义中需要添加修改错误类型2为:函数调用时未经声明或者定义这样的字眼,更加严谨一点;**

4. 我大部分的程序是根据产生式书写的递归形式的,比如ExtDef_s(...){xxx;type a=Specifier_s(...);FunDec_s(..a..);...}这种,略过不谈;我细说一下我用循环实现的structspecifier和vardec里面的数组部分;

   之所以使用循环是因为struct结构中需要进行额外的链表链接操作,和其他单独定义的变量不一样，缺点是代码可读性的降低和难以debug;这个循环结构我主要通过strcmp确定处于产生式的位置,然后通过两个FieldList类型的指针result,tempfield进行链表的获取,记录,链接,最后给struct的structure元素result这个指针;

   struct的遍历是顺序和产生式一致的,在vardec中的遍历顺序和要链接的链表顺序是相反的,具体而言对于$a[10][3][2]$访问的顺序是2->3->10->ID,但是要组成的是ID->10->3->2->INT这样的链表结构,因此我的方法是首先进行一次遍历获得深度,然后根据这个深度malloc一个type_list数组,然后再遍历一次填写数组,最后根据这个数组组装成链表;

5. 主要的bug:
   1. 我的strcmp可能有3-5次忘记写==,给我的警醒是以后写代码的时候要么宏定义,要么使用$0 == strcmp()$这样的结构;
   2. 一些理解的疏漏,我因此更新了好多次query和insert函数,用于获得更多的信息;
   3. 我在编写代码的时候为了防止疏漏加了很多assert(0);因此经常会有一些"漏网之鱼",触发assert(0);这种方法找到了很多报错了之后没有即使return的bug;
6. 感觉可以还可以改进的地方:
   1. 使用循环的地方可能有一些冗余,我感觉array处理的链表扫了3次不是那么高效,也许可以只扫两次;
   2. 代码太长了一点,我零零碎碎包括注释加起来2k3左右,实在是太不简洁了;

编译方法

1. 使用make 编译，make test测试相应test文件夹的对应文件，或者./parser test.cmm 测试

反馈

1. 再一次感谢同学们一起出的测试,帮助很多!
2. 讲义的一些问题写的不是很清楚，在群里也有讨论，如果可能的话下一版把这些讨论之后的案例加进去，这样讲义就更严谨了!