编译原理第二次实验报告

201220069 周心同

实验内容

实现内容：包含所有的必做要求和选做要求并通过了所有oj样例。

本次实验主要是在原先词法分析和语法分析的基础上进行语义分析，具体内容是将c源代码对应的错误类型和行数打印出来，具体操作就是遍历实验一得到的语法树然后分各种各样的情况讨论，同时维护一个符号表便于联系上下文。

该实验属于难度不大，但是繁杂琐碎，需要考虑的地方很多，个人觉得非常非常麻烦。

编译方式

实验环境：与实验要求相同。

在Code目录下make即可。

程序亮点

1.散列表

按照实验推荐的散列函数和闭散列方法实现，效率较高。

2.作用域

参考了实验指导书推荐的十字链表，但没有使用，原因在于十字链表要用到交叉的指针，实现难度大，这对于本实验来说很可能会造成难以解决的bug，在这里我用了一种比较简单高效的方法。

构造一个新的数据类型：作用域，并定义一些全局变量，如下：

```
//作用域
typedef struct scope{
    int id;
    int parent_id;//自己外面一层的作用域
    int wno;//自己和自己所有外层作用域的数量
    int w[50];//自己和自己所有外层作用域的id
}scope;
int scope_id = 0;//生成新作用域的id
int current_id = 0;//当前作用域
scope sc_table[50];
```

我们对作用域的用处有两个地方，一个是看该变量名能不能被定义，也就是说该名字有没有在同层下重复定义过或者与结构体类型相同，

此时就需要当前的作用域信息，这里我用current_id标记当前作用域，sc_table则是一个作用域(散列)表，这样就能通过current_id找到当前作用域。

另一个是使用变量时看先前有没有被定义，这就需要从里层往外层找变量的定义，此时就需要知道当前作用域和其外层作用域的范围，故而该结构需要知道wno和w。其中计算其外层作用域范围的方法只需要用其parent作用域的w加上自己即可。（也可只往parent遍历）

同时我们还需要进行进入新的作用域和退出当前作用域的操作，而退出当前作用域也需要parent_id的存在。

这样做也还有一个好处就在于，退出作用域不会删去这个作用域的变量，只是在语义分析的判断中不会用到，但可能会在未来的分析中起到作用。

实验建议和感想

1.实验指导书上写的数据结构kind中应该加一项结构体类型，或者将结构体类型单列出来，因为结构体类型和结构体变量是不一样的，而如果把结构体类型也放入符号表中，它们的地位是不同的：新定义的变量不能与结构体类型名同名，哪怕是在不同作用域。

对此我的做法是，首先在OptTag处，这是结构体类型名产生的唯一地方，标记它为STR_SPE类型，其次在Specifier调用StructSpecifier时，返回时加一句type->kind =STRUCTURE;这是结构体变量的类型产生的地方。然后在对符号表操作时加一些特殊标记。

2.实验非常容易出bug，难点也在此，所以debug真的很重要，这里我主要通过printf和第一次实验的printtree在对程序进行观察，可以说实验一的print结果非常有帮助。这个实验也非常需要大局观，开始的时候就应该注意到应用作用域的两个地方，结构体类型，函数定义和声明，函数名和变量名可以重复等等问题，否则后面debug非常困难。

一些问题的反思

Segmentation fault：往往是NULL指针出了问题，主要在于对NULL的访问，深层原因可能是缺少if判断，结点的child索引错了以及一些理解上的bug。

malloc.c:2379: sysmalloc: Assertion `(old_top == initial_top (av) && old_siz：注意malloc分配大小应该是sizeof(Type_)而不是sizeof(Type)，Fieldlist同理，原因在于

```
typedef struct Type_ *Type;
typedef struct FieldList_ *FieldList;
```

一些奇怪的打印错误：主要检查符号表的操作和返回值问题，有可能在某个函数里面if判断后忘了返回，或者忘了将变量加入符号表等。