编译原理实验二报告

在实验二中，我们需要在实验一词法分析和语法分析程序的基础上编写一个程序，对C源代码进行语义分析和类型检查，并打印分析结果。

符号表的实现

为了实现语义分析诸如检查变量类型以及重复定义等功能，我们需要构建一个符号表进行填表和查表工作。我们组的选做是3.1，不需要处理变量的定义域，即所有变量都具有全局定义域，这极大简化了符号表的实现。我选择使用散列表作为符号表，由于类型与变量不可重名，可以考虑将它们都放在一张散列表里。

为了避免冲突，可以采用开放寻址法，如果重复则增加一个偏移量知道没有重复。由于我们无需进行删除操作，读取时按照索引以及偏移量向后遍历，直至名称与输入匹配则为找到，如果遇到了空指针则说明不存在。

散列表的每个元素是一个sem_node, type参考了示例中的实现，func_dec_lineno是为了记录函数声明的行号，便于发现声明未定义错误。

```
typedef struct sem_node {
    char *name;
    Type type;
    int func_dec_lineno;
} sem_node;
```

填表部分

需要填表的内容包括函数、结构体、变量的定义。它们都位于ExtDef下。这一部分的实现类似于暴力蛮干，我实现了一些read系列函数，根据节点类型，使用对应的read函数便可以进行分析并得到结果。

ExtDef

首先，为了识别3.1要求的函数声明，我们需要增加一条规则：Specifier FunDec SEMI

根据语法规则，ExtDef主要由一个Specifier和一个ExtDecList或者FunDec组成。因此我们需要三个对应的read函数。

1. Specifier可以是基本类型(int/float)或者结构体定义。前者十分简单，对于后者，则需要区分两种情况，即使用之前定义的struct以及声明一个新的struct，前者查表即可，十分简单。struct可能匿名，这里我为其分配了一个匿名struct专有的独特名称来解决这个问题。读取新struct定义时，可以按照其内部的DefList依次添加FieldList到struct中field的tail。
2. ExtDecList 主要包含一些变量的名称。在这里要特别注意数组的情况，此时不能简单地把当前变量的type设置为Specifier的type，而需要创建一个array，将type设置为array的elem。多维数组如法炮制即可。

3. FunDec 为函数定义，对于函数参数VarList，可以按照类似ExtDecList 的方法添加进函数的field。这里需要注意函数的返回值，我的实现中函数的返回值会被添加到函数的field的末尾。

注意点：

- DecList出现在函数的CompSt和struct的DefList中。函数的CompSt中是允许赋值语句的，而struct不允许，需要进行区分。同时，在struct中的变量重复应该算作error 15，而非error 3

查表部分

需要查表的部分基本都出现在Exp中。Exp出现的位置比较多，且大多相对独立，无需上层结构传递信息便可完成分析(对函数返回值的检查除外)。因此，在Exp的分析中，我采取了对子树前序遍历，直至找到Exp节点并对其分析，找到后可直接跳过其子节点，直接到其兄弟节点，这样就避免了重复。

Exp

read_Exp的返回值是一个Exp_type, 其中type表示语句的type,l_val表示该语句是否是左值, error表示错误，这主要是为了避免连锁反应，如果检测到error说明已经进行了报错，无需再次报错。Exp大概分为几种情况，下面我将分开来叙述。

```
typedef struct Exp_Type {
    Type type;
    int l_val;
    int error;
} Exp_Type;
```

INT/FLOAT/ID

这是最简单的部分，ID说明为变量，查表返回对应的type即可。注意这里的ID是左值。

ID LP Args RP/ID LP RP

这是函数调用，查表找到函数的返回值，返回即可。同时需要注意检测Args是否符合函数定义时的参数要求。

Exp LB Exp RB

这是数组访问，可以递归查询Exp的type，然后返回elem即可。注意这里也是左值。

Exp DOT ID

这是结构体访问，可以递归查询Exp的type，然后返回对应域的type即可。这里也是左值。

MINUS Exp/LP Exp RP/Exp DIV Exp/Exp STAR Exp/Exp MINUS Exp/Exp PLUS Exp

这些表达式基本相同，返回值都是子Exp的type。这里需要注意检测运算符两侧的Exp的type是否相同，且满足一定必要的条件，比如只有int和float才能参与算数表达式。

Exp OR Exp/NOT Exp/Exp AND Exp/Exp RELOP Exp

这里是逻辑表达式，返回值是int，运算符两侧的Exp的type必须相同且为int。

Exp ASSIGNOP Exp

赋值语句两侧的Exp的type必须相等，同时左侧的Exp需要检查是否是左值。注意返回值是Exp的type！因为可能有连等式的存在。

检查函数的返回值

这是Exp中为数不多的需要传递参数的地方。我选择在read_CompSt中对子树进行扫描时检查节点是否是RETURN，如果是就检查下面的Exp的type是否对应。这里需要注意函数可能有多分支，所以return可能不止一个。

如何运行

已经按照oj的要求打包好了，使用make便可编译运行。