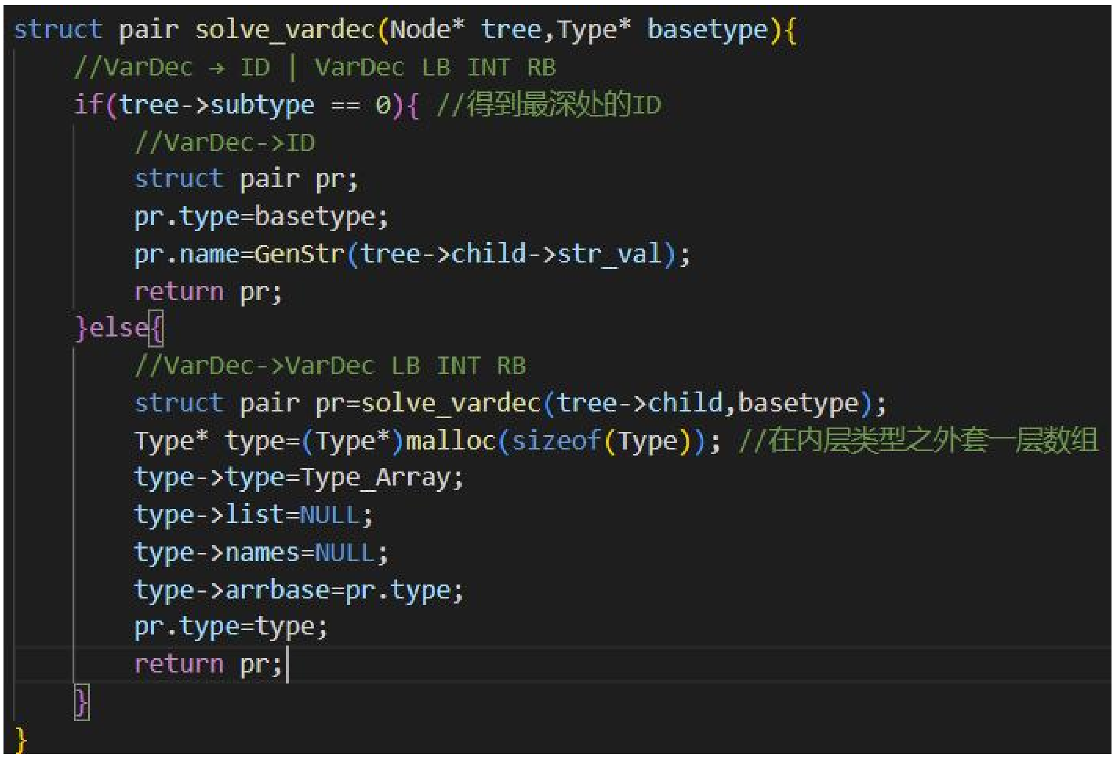
实验二、语义分析

姓名：瞿久尧　　　　　　　　　　　学号：120L022314

1　功能介绍及实现

1.1 solve_vardec

递归函数，递归解析用于定义变量的 VarDec，主要是要把数组类型以及变量名称给解析出来



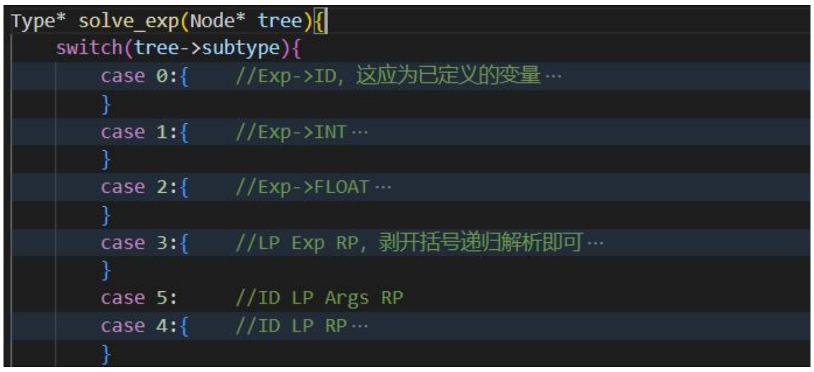
```
struct pair solve_vardec(Node* tree,Type* basetype){
    //VarDec → ID | VarDec LB INT RB
    if(tree->subtype == 0){ //得到最深处的ID
        //VarDec->ID
        struct pair pr;
        pr.type=basetype;
        pr.name=GenStr(tree->child->str_val);
        return pr;
    }else{
        //VarDec->VarDec LB INT RB
        struct pair pr=solve_vardec(tree->child,basetype);
        Type* type=(Type*)malloc(sizeof(Type)); //在内层类型之外套一层数组
        type->type=Type_Array;
        type->list=NULL;
        type->names=NULL;
        type->arrbase=pr.type;
        pr.type=type;
        return pr;
    }
}
```

1.2 solve_exp

递归函数，对 Exp 进行递归解析，通过 subtype 得知 Exp 的产生式。

实现方式

对于一个 Exp 结点及其子树，我们只需要通过递归得到两个综合属性，一个是表达式值的类型。另外一个就是该表达式是否只有右值（表达式不能被赋值，不能被放在赋值运算符的左侧）。



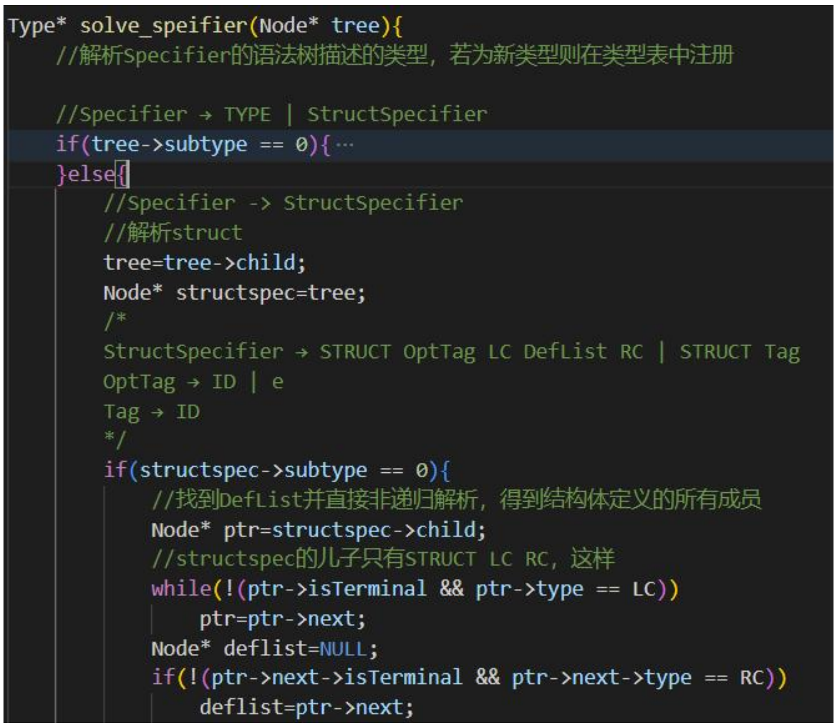
```
Type* solve_exp(Node* tree){
    switch(tree->subtype){
        case 0:{    //Exp->ID，这应为已定义的变量 ···
        }
        case 1:{    //Exp->INT ···
        }
        case 2:{    //Exp->FLOAT ···
        }
        case 3:{    //LP Exp RP，剥开括号递归解析即可 ···
        }
        case 5:     //ID LP Args RP
        case 4:{    //ID LP RP ···
        }
```

1.3 solve_speifier

递归函数，专门对 Specifier 进行解析，返回对应的类型 Type*

实现方式

在定义函数或变量时，根据 Specifier 可以得知类型，如果是基本类型如 INT 和 FLOAT 则查类型表，如果是一个新的结构体定义，那么就要处理结构体内各个成员的定义，提取出各个成员的名字，类型。



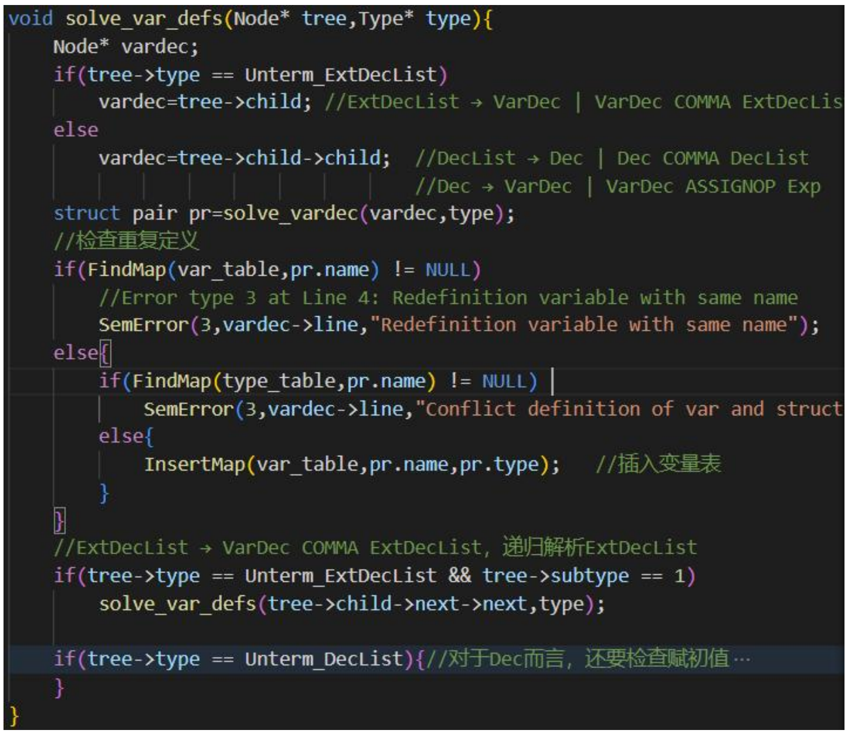
```
Type* solve_speifier(Node* tree){
    //解析Specifier的语法树描述的类型，若为新类型则在类型表中注册

    //Specifier → TYPE | StructSpecifier
    if(tree->subtype == 0){···
    }else{
        //Specifier -> StructSpecifier
        //解析struct
        tree=tree->child;
        Node* structspec=tree;
        /*
        StructSpecifier → STRUCT OptTag LC DefList RC | STRUCT Tag
        OptTag → ID | e
        Tag → ID
        */
        if(structspec->subtype == 0){
            //找到DefList并直接非递归解析，得到结构体定义的所有成员
            Node* ptr=structspec->child;
            //structspec的儿子只有STRUCT LC RC，这样
            while(!(ptr->isTerminal && ptr->type == LC))
                ptr=ptr->next;
            Node* deflist=NULL;
            if(!(ptr->next->isTerminal && ptr->next->type == RC))
                deflist=ptr->next;
```

1.4 solve_var_defs

递归函数，递归解析用于定义变量的 VarDec，主要是要把数组类型以及变量名称给解析出来。

实现方式

DecList 和 ExtDecList 都可以视作 VarDec 的列表，只需要一个一个一个地拆出 VarDec，然后使用 solve_vardec 解析，得到新定义的变量名字与类型，然后把它们在变量表中注册即可。



```c
void solve_var_defs(Node* tree,Type* type){
    Node* vardec;
    if(tree->type == Unterm_ExtDecList)
        vardec=tree->child; //ExtDecList → VarDec | VarDec COMMA ExtDecLis
    else
        vardec=tree->child->child;  //DecList → Dec | Dec COMMA DecList
                                    //Dec → VarDec | VarDec ASSIGNOP Exp
    struct pair pr=solve_vardec(vardec,type);
    //检查重复定义
    if(FindMap(var_table,pr.name) != NULL)
        //Error type 3 at Line 4: Redefinition variable with same name
        SemError(3,vardec->line,"Redefinition variable with same name");
    else{
        if(FindMap(type_table,pr.name) != NULL)
            SemError(3,vardec->line,"Conflict definition of var and struct
        else{
            InsertMap(var_table,pr.name,pr.type);   //插入变量表
        }
    }
    //ExtDecList → VarDec COMMA ExtDecList, 递归解析ExtDecList
    if(tree->type == Unterm_ExtDecList && tree->subtype == 1)
        solve_var_defs(tree->child->next->next,type);

    if(tree->type == Unterm_DecList){//对于Dec而言，还要检查赋初值…
    }
}
```

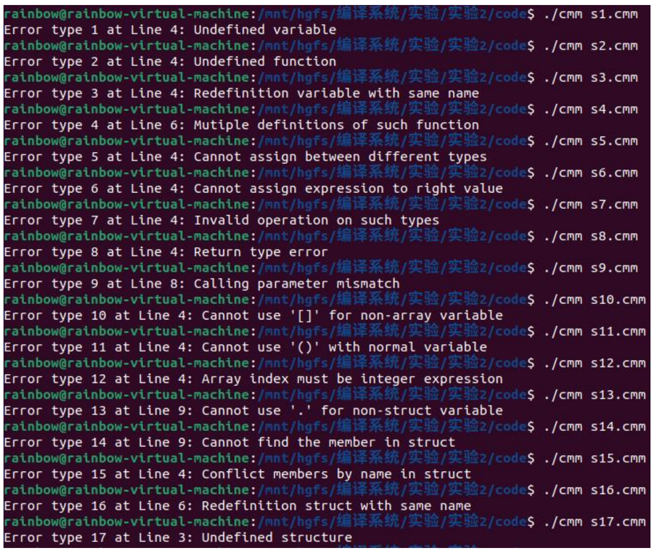
2 编译过程

在文件夹中打开终端，输出./make.sh 得到可执行文件 cmm.o(压缩包中已包含 cmm.o,这一步也可以省略)

对于纠错 s1.cmm,输入./cmm s1.cmm，纠出 s1 的错误 Error type 1 at Line 4: Undefined variable;s2 到 s17 的纠错同理

3 测试结果

本程序对指导书中的 17 个样例均进行了测试并获得了正确的结果



```
rainbow@rainbow-virtual-machine:/mnt/hgfs/编译系统/实验/实验2/code$ ./cmm s1.cmm
Error type 1 at Line 4: Undefined variable
rainbow@rainbow-virtual-machine:/mnt/hgfs/编译系统/实验/实验2/code$ ./cmm s2.cmm
Error type 2 at Line 4: Undefined function
rainbow@rainbow-virtual-machine:/mnt/hgfs/编译系统/实验/实验2/code$ ./cmm s3.cmm
Error type 3 at Line 4: Redefinition variable with same name
rainbow@rainbow-virtual-machine:/mnt/hgfs/编译系统/实验/实验2/code$ ./cmm s4.cmm
Error type 4 at Line 6: Mutiple definitions of such function
rainbow@rainbow-virtual-machine:/mnt/hgfs/编译系统/实验/实验2/code$ ./cmm s5.cmm
Error type 5 at Line 4: Cannot assign between different types
rainbow@rainbow-virtual-machine:/mnt/hgfs/编译系统/实验/实验2/code$ ./cmm s6.cmm
Error type 6 at Line 4: Cannot assign expression to right value
rainbow@rainbow-virtual-machine:/mnt/hgfs/编译系统/实验/实验2/code$ ./cmm s7.cmm
Error type 7 at Line 4: Invalid operation on such types
rainbow@rainbow-virtual-machine:/mnt/hgfs/编译系统/实验/实验2/code$ ./cmm s8.cmm
Error type 8 at Line 4: Return type error
rainbow@rainbow-virtual-machine:/mnt/hgfs/编译系统/实验/实验2/code$ ./cmm s9.cmm
Error type 9 at Line 8: Calling parameter mismatch
rainbow@rainbow-virtual-machine:/mnt/hgfs/编译系统/实验/实验2/code$ ./cmm s10.cmm
Error type 10 at Line 4: Cannot use '[]' for non-array variable
rainbow@rainbow-virtual-machine:/mnt/hgfs/编译系统/实验/实验2/code$ ./cmm s11.cmm
Error type 11 at Line 4: Cannot use '()' with normal variable
rainbow@rainbow-virtual-machine:/mnt/hgfs/编译系统/实验/实验2/code$ ./cmm s12.cmm
Error type 12 at Line 4: Array index must be integer expression
rainbow@rainbow-virtual-machine:/mnt/hgfs/编译系统/实验/实验2/code$ ./cmm s13.cmm
Error type 13 at Line 9: Cannot use '.' for non-struct variable
rainbow@rainbow-virtual-machine:/mnt/hgfs/编译系统/实验/实验2/code$ ./cmm s14.cmm
Error type 14 at Line 9: Cannot find the member in struct
rainbow@rainbow-virtual-machine:/mnt/hgfs/编译系统/实验/实验2/code$ ./cmm s15.cmm
Error type 15 at Line 4: Conflict members by name in struct
rainbow@rainbow-virtual-machine:/mnt/hgfs/编译系统/实验/实验2/code$ ./cmm s16.cmm
Error type 16 at Line 6: Redefinition struct with same name
rainbow@rainbow-virtual-machine:/mnt/hgfs/编译系统/实验/实验2/code$ ./cmm s17.cmm
Error type 17 at Line 3: Undefined structure
```