L3实验报告——代码REVIEW

实现的功能：

- 在词法分析和语法分析基础上对C--源码进行语义分析和类型检查，打印分析结果
- 若存在语义错误，打印错误信息
- 若不存在语义错误，不进行输出

实现方法：

- 在词法分析和语法分析的基础上，添加语义分析代码
- 头文件semantic.h
  - 包含由实验指导中提供的类型表示方法，并在TYPE中添加函数类型：函数类型的信息包括参数链表，函数返回值类型，参数个数
  - 定义各种函数，具体实现在semantic.c
- 具体实现semantic.c
  - 基本思想：循环+递归
  - 符号表：使用实验指导中提供的散列表hash函数
  - int insertSymbol(FieldList f)：插入符号，用hash函数找到要插入的位置，如果该位置为空就直接插入，不为空则向下找到一个空的位置插入
  - FieldList findSymbol(char* name, int function)：在符号表中寻找符号，用hash函数找到位置，匹配name是否相同，如果不同则向下寻找，找到返回该符号的FieldList，没有找到返回NULL，function参数为函数标识，如果要找的是函数则为1，否则为0，防止出现变量名和函数名相同的情况下返回错误的FieldList
  - int TypeEqual(Type type1, Type type2)：比较类型是否相等
    - 对于basic，直接比较basic　type1->u.basic == type2->u.basic
    - 对于array，递归比较元素类型　　TypeEqual(type1->u.array.elem, type2->u.array.elem)
    - 对于structure，比较其每个域的类型是否一样
    - 对于函数，先比较参数个数，相同的情况下逐个比较参数域的类型
  - 根据c--文法，从ExtDefList切入开始处理语法树
  - FieldList VarDec(Node* root, Type basicType)：构建数字或数组的FieldList并插入到符号表中(如果没有错误)，并返回该FieldList
    - 如果root不是ID，则说明是数组，使用while循环得到维数，使用for循环构建数组
    - 如果root是ID，则说明是数字
  - Type Specifier(Node* root)：识别int、float、struct,并同时将这些符号插入到符号表中（如果没有错误），返回识别到的类型
    - Specifier->TYPE：int或者float
    - Specifier->StructSpecifier：structure
      - StructSpecifier->STRUCT Tag：使用findSymbol找Tag的FieldList，如果没找到则说明struct没有定义过，报错，否则返回该struct type
      - StructSpecifier->STRUCT OptTag LC DefList RC：使用while循环处理DefList，循环内部用while循环处理DecList，处理的过程中生成struct内部域的tail链，并将每个域插入到符号表中(如果没有错误)，处理完后看OptTag是否存在，存在就在符号表中找，如果找到就说明重复定义，不存在就插入到符号表中

    - void CompSt(Node *root,Type returnType)：CompSt->LC DefList StmtList RC，调用DefList(CompSt->child[1])和循环调用Stmt(Stmt_,returnType)处理CompSt
    - void DefList(Node *root)：while循环处理并构建符号域并插入到符号表
    - void Stmt(Node *root,Type returnType)：处理Stmt，对Stmt的每个产生式单独处理
    - Type Exp(Node* root)：处理Exp，返回类型
        - 根据Exp的产生式，可以合并一部分可以同时处理的Exp
            - Exp->LP Exp RP　Exp->MINUS Exp　Exp->NOT Exp
            - Exp->Exp PLUS Exp　Exp->Exp MINUS Exp　Exp->Exp STAR Exp　Exp->Exp DIV Exp
                - 数字运算，需要比较操作符两边的操作数类型是否匹配，返回的类型为其中一个的类型
            - Exp->Exp AND Exp　Exp->Exp OR Exp　Exp->Exp RELOP Exp
                - 逻辑运算，需要比较操作符两边的操作数类型是否匹配，返回值为INT
        - Exp->ID：符号表中找，找到返回该符号的type，没找到就是没定义
        - Exp->INT　Exp->FLOAT：单个数字，构建对应的type并返回
        - Exp->Exp ASSIGNOP Exp
            - 这里先要看第一个Exp是否是右值，如果是右值，报错并返回NULL
            - 如果是左值，比较两个Exp类型是否相同，不同返回NULL，相同返回其中一个的type
        - Exp->ID LP RP　Exp->ID LP Args RP：
            - 函数调用，先找有没有该函数，没有则报错并返回NULL
            - 如果有Args，构建Args的tail链
            - 比较函数的type和调用时的type，不同则报错并返回NULL
        - Exp->Exp DOT ID
            - 结构体内部域调用，先看该Exp是否是结构体，如果不是，报错并返回NULL
            - 如果是，while循环比较struct内部域中是否有匹配的name，如果没有，报错并返回NULL
            - 返回该匹配域的type
        - Exp->Exp LB Exp RB
            - 数组使用，同样的，先看Exp是否是数组，如果不是，报错并返回NULL
            - 如果是，看[]中间的参数是不是int类型，如果不是，报错并返回NULL
            - 返回该域的类型
- 多维数组和结构体的类型标识使用实验指导中提供的类型表示

有趣现象

- 一开始写完的时候，提交会出现各种runtime error，使用计科的代码进行本地debug的时候就会发现是空指针的问题

一些bug

- 最多的还是runtime error，空指针的问题，NULL->XXX或者说没给内存空间
- int a = a+1不需要报undefined variable错误
- 多维数组的构建bug
- 结构体和函数定义时的tail链构建bug
- 右值判断不全，比如少判断了负数
- (计科测试用例)超32位id
- normal20多报？(到现在我都不知道哪有问题)