实验 2　　191220154　　张涵之

a) 实现功能
   a) 必做内容（检查语义错误类型 1-17）；
   b) 选做内容（要求 2.3）；

b) 实现思路
   a) 在实验 1 中，语法树相关数据结构的定义放在 syntax.y 中，而实验二需要对语法树进行遍历分析，这样的定义方法有诸多不便，故将语法和语义相关数据结构和函数移至 lib.h 和 lib.c 中，在 semantic.h 和 semantic.c 中实现具体的语义分析。
   b) 定义 Type_表示数据类型，有数（整数和浮点数）、数组、结构体和函数几种基本类型，同时定义了错误类型，在 u 中以错误码标识具体错误种类。
      - 数的具体类型由 u 中的 basic 指定，分为 BASIC_INT 和 BASIC_FLOAT；
      - 数组由数组元素的类型和数组长度指定，这样可以递归地定义多维数组；
      - 结构体由结构体内部定义的域形成的一个链表来表示；
      - 函数由定义的 struct function 表示，包括函数返回值和参数列表；

```
struct Type_ {
    enum { BASIC, ARRAY, STRUCTURE, FUNCTION, ERROR } kind;
    union {
        int basic;
        struct { Type elem; int size; } array;
        FieldList structure;
        struct { Type returnType; FieldList parameters; } function;
        int errorCode;
    } u;
};

struct FieldList_ {
    char name[32];
    Type type;
    int defineLine;
    FieldList tail;
};

struct Symbol_ {
    char name[32];
    Type type;
    int defineLine;
    Symbol hashTail;
};
```

   c) Symbol_表示符号表中的元素，FieldList_类似，相当于局部“小符号表”，用于表示函数定义这的参数列表、结构体内部的域列表，包含元素的名字、类型，初次被定义并加入表中的行号，以及指向表中下一个节点的指针。
   d) 考虑局部列表（函数参数、结构体域）一般不会太大，直接用单向链表实现，全局符号表采用手册中介绍的散列表结构，采取 open hashing 解决冲突问题。
   e) 提供了相关函数进行 Type_，FieldList_和 Symbol 的定义和比较。
   f) 语义分析具体实现部分采取语法分析和语义分析分开的方式，从根节点开始对语法树进行遍历处理，对应产生式中的各种非终结符号定义处理函数，根据产生式相互调用，其中如 DefList，Def，DecList，Dec 另外增加了针对结构体中局部变量定义的处理函数，因为在其他情境下（如函数内部的局部变量定义），函数只需查符号表并根据查表结果进行插入或报错即可，不需要返回任何值，而对于结构体，前面设计的表示方法还需要返回一个 FieldList，从而对其进行命名“查重”工作。

```
// High-level Definitions
void Program(Node* cur);
void ExtDefList(Node* cur);
void ExtDef(Node* cur);
void ExtDecList(Node* cur, Type extDecType);
// Specifiers
Type Specifier(Node* cur);
Type StructSpecifier(Node* cur);
char* OptTag(Node* cur);
Type Tag(Node* cur);
// Declarators
Symbol VarDec(Node* cur, Type varDecType);
Symbol StructVarDec(Node* cur, Type varDecType);
void FunDec(Node* cur, Type funcDecType);
FieldList VarList(Node* cur);
FieldList ParamDec(Node* cur);
// Statements
void CompSt(Node* cur, Type compStType);
void StmtList(Node* cur, Type stmtType);
void Stmt(Node* cur, Type stmtType);
// Local Definitions
void DefList(Node* cur);
FieldList StructDefList(Node* cur);
void Def(Node* cur);
FieldList StructDef(Node* cur);
void DecList(Node* cur, Type decType);
FieldList StructDecList(Node* cur, Type decType);
void Dec(Node* cur, Type decType);
FieldList StructDec(Node* cur, Type decType);
// Expressions
Type Exp(Node* cur);
FieldList Args(Node* cur);
```

g) 错误类型的具体发现和处理：

- 与未定义和重复定义有关的（1、2、3、4、16、17）查符号表即可发现；
- 类型不匹配有关的（5、6、7、8、9、10、11、12、13）比较 Type_可发现；
- 错误类型 14：遍历结构体的 FieldList 列表查找域的名字即可发现；
- 错误类型 15：对 StructDefList 返回的 FieldList 进行两重循环遍历即可发现；
- 要求 2.3：将比较结构体名字改为比较 FieldList 中每个域的 Type_即可；
- 错误的处理：起初多处采用 return NULL，后发现会造成意想不到的段错误，如各非终结元素对应函数相互调用时，很容易造成对空指针的错误操作，因此尽量少使用 NULL 而改为返回 errorType，或者设置特定的标志（如 name 设为“123”，则它不可能与任何非终结符或 ID 名重复，以此作为调用过程中出现错误的信号。

c) 反思与总结

a) 先前 Node 的定义中对 type_int，type_float 和 type_str 进行 union 定义，发现并无必要且对输出造成很大麻烦，由 str 转 int float 易，由 int float 入 str 难，因此将联合类型改为 char*，后续计算时可现场进行 atoi、atof 操作，更方便。

b) 输出调试信息的时候，如果类型相对比较复杂（如样例 9 中参数类型如果是数组或者结构体），则很难输出具体和正确的名称，由于时间有限，这里没有进行很好的处理，只是输出了“Function (funcname) is not applicable for arguments”。

d) 编译运行

使用 makefile 进行编译。直接 cd 到 Code 目录下 make 即可。

编译完成后，输入命令./parser test 即可对 test 文件进行分析。