实验2 191220154 张涵之

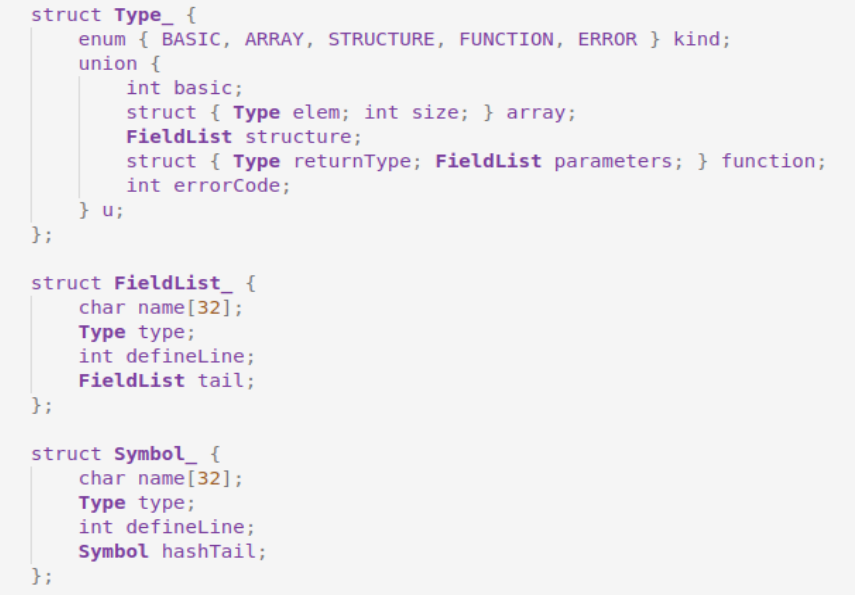
1. 实现功能
   1. 必做内容（检查语义错误类型1-17）；
   2. 选做内容（要求2.3）；
2. 实现思路
   1. 在实验1中，语法树相关数据结构的定义放在syntax.y中，而实验二需要对语法树进行遍历分析，这样的定义方法有诸多不便，故将语法和语义相关数据结构和函数移至lib.h和lib.c中，在semantic.h和semantic.c中实现具体的语义分析。
   2. 定义Type\_表示数据类型，有数（整数和浮点数）、数组、结构体和函数几种基本类型，同时定义了错误类型，在u中以错误码标识具体错误种类。

- 数的具体类型由u中的basic指定，分为BASIC\_INT和BASIC\_FLOAT；

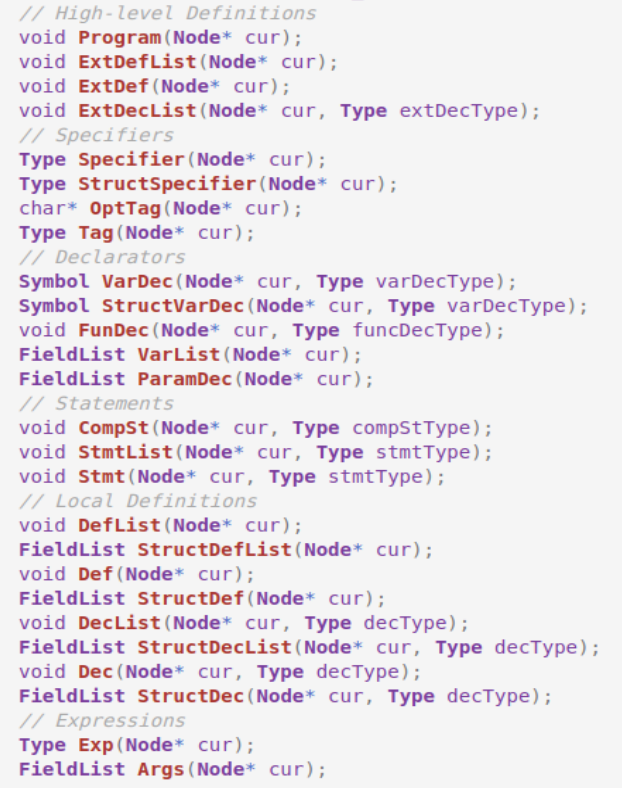
- 数组由数组元素的类型和数组长度指定，这样可以递归地定义多维数组；

- 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；



* 1. 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。
  2. 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。
  3. 提供了相关函数进行Type\_，FieldList\_和Symbol的定义和比较。
  4. 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。



* 1. 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。

1. 反思与总结
   1. 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。
   2. 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”。
2. 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。