编译原理Lab2实验报告

1. 完成情况：本次实验完成了所有选做要求，通过了OJ所有测试。
2. 编译方法：`make`生成可执行文件; `./parser <filename>` 运行测试用例。
3. 因为实验使用的是C语言，构造符号表需要手写很多的数据结构（C++ STL用惯了，感觉非常痛苦），因此我选择了先实现hash表。实现的时候，我选择了直接实现Imperative Style样式的符号表（其他数据结构就默认照着讲义来写了）
4. 实现hash表之后，就开始实现整个语法分析过程，实现方式也很简单，对于每个不同的节点，采取不同的语法制导翻译规则。这一部分给我的感受就是设计一个良好易读的架构非常的重要。比如说StructSpecifier这一部分，结构体的类型就得从DefList中获取。同时可以发现，DefList内部的属性只对StructSpecifier有用（从其他地方推导出DefList是不会影响父节点的），于是我们可以让DefList返回一个FieldList来为这个Struct类型初始化。DefList推导出Def DefList, 因此Def的返回值也是FieldList. Def又推导出Specifier DecList SEMI, 那么我们让Def直接返回DecList的结果（当然，这里得先遍历Specifier获取基础类型，传给DecList），然后遍历DefList -> Def DefList中推导出的DefList，将两段链表再拼接起来，就得到了一个完整的FieldList链，可以初始化为新的Struct类型中的u.structure字段。
5. （感觉实现也没啥好讲的就讲下附加功能的实现吧）
    1. 2.1的错误判断有一点比较特殊，那就是这个错误不是一遍遍历能当场发现的（声明而未定义函数），因此我在符号表中新开了一个链表，每次插入函数的时候，如果成功插入，就去更新链表，链表中维护了所有的已声明/定义函数，并且维护是否声明，所在行号的信息。在Program遍历完毕的时候，调用检查函数判断是否存在声明而未定义的函数，存在就打印错误信息。
    2. 2.2的实现就是使用stack维护每一层的符号，每次进入一层就更新stack，弹出一层就从符号表中删除stack首部指向的一串符号。这里可以证明，如果每次插入表头，stack链也每次插入表头，那么被删除的元素一定是hash表指向的第一个元素，因此删除操作就变得简单了很多。
    3. 2.3的实现，我使用了一个函数，传入两个fieldlist, 用两个迭代指针指向fieldlist并遍历，对比每个fieldlist中的type, 在调用type类型比较之后，如果不相等，那么返回值就是不相等，否则迭代到fieldlist->tail接着比较。这样子就能实现结构等价。（当然，必须实现其他的类型(BASIC, ARRAY)等价函数，这样这些等价函数就可以互相调用）
6. 总结：这次实现的代码量确实挺大的，而且如果设计不当，会极大地影响代码效率。同时由于涉及到许多的malloc内存管理，调试也会变得很困难。我在调试的过程中添加了 -fsanitize=address 参数, 能够报告内存泄漏（虽然已经不在乎了）和访问越界的错误，还能提供调用栈的信息，非常好用。