191098328 计算机科学与技术系 张世茂

邮箱：191098328@smail.nju.edu.cn

在本次实验中，我在实验一已经完成的词法分析和语法分析程序的基础上，通过对已经生成的语法树进行遍历对特定的 C--语言源程序文件完成进一步的语义分析，并在程序中出现目标错误时及时查出并报出对应的错误信息，在没有错误时正常退出程序没有输出。此外，针对附加任务 2.1 的要求，我还实现了对于函数声明的语法检测及相关的错误检测。

要实现上述上述提到的功能，对程序进行正确的语法分析，首先是基于对语法树的遍历。遍历的过程中要注意到一些关键节点。首先由于整个程序本质上都是由多个 ExtDef 节点产生的，而 ExtDef 节点本身就代表产生了一个对于变量或函数的定义，且由 Program 节点到此的路径不需要进行过多关心，因此在遍历函数中只在每次遇到 ExtDef 节点时调用同名的处理函数即可，其余情况下继续向下递归遍历其子节点，对以 ExtDef 节点为根的子树而言，其处理和遍历将在处理函数中完成而无需在遍历函数中进行。

实验的两个要点分别是建立符号表和检查程序错误。对于符号表，在本实验中根据需要我总共建立了 6 个不同的符号表来分别按类存储所需要的信息，具体如下：

```
VarNode* VarTable = NULL;

ArrNode* ArrTable = NULL;

StructNode* StructTable = NULL;

StructTypeNode* StructTypeTable = NULL;

FuncNode* FuncTable = NULL;

UndefinedFuncNode* UndefinedFuncTable = NULL;
```

其中 VarTable 存储了 int 类型和 float 类型的普通变量，ArrTable 存储了定义的数组类型变量，StructTable 存储了程序中定义的结构体变量，StructTypeTable 存储了定义的结构体类型名，FuncTable 存储了程序中定义或声明过的函数，由于涉及到函数已声明但未定义的情况，通过遍历 FuncTable 逐一判断标志位又较为麻烦，且一个函数可以有多次声明的相关信息被存储，因此单独定义了一张表 UndefinedFuncTable 来暂时存储那些声明时尚未定义的函数声明的信息。并为每个表分别设计函数进行表项的插入。

普通类型的数据便于表示和存储，主要的难点在于实现对数组和结构体有关的表示存储，因为他们不仅本身具有相对复杂的结构，且涉及到嵌套定义等一系列的问题。因此选用链表解决这一问题。对每个结构体类型设定一个 FieldList 的域作为其定义中所包含的域的链

表的头指针，每个节点中都存储了一个域的相关信息。而对于数组的类型，由于涉及到多维数组，因此仍然用一个链表，中间的每一维存储类型标识为 ARRAY，最后一维存储数组中存储的数据的基本类型，也即每一个链表节点中都表示了当前这一维数组中存储的数据类型。

在处理语法单元的语义信息的过程中，为了提高代码的简洁性和功能间互相调用的方便，对大多数语法单元的处理都进行了同名函数的封装，还对一些譬如类型判断，类型查找之类的常用功能进行了封装。涉及到一些细节上的实现方法需要结合功能需要进行实际上的统筹考虑，例如在实现 VarDec 节点的同名函数时，由于 VarDec 可能是维数未知的多维数组，涉及到与外层类型嵌套表示的问题，因此将 VarDec 的类型生成单独作为一个函数进行实现。由于在实现过程中用到了大量的链表操作，这就带来了大量诸如野指针访问之类的问题，极易在未知的情况下导致程序出错崩溃，因此在实现的过程中也可以在适当的特殊位置添加一些 assert 断言来在程序出错时辅助排查。

关于包括必做部分和选做 2.1 要求的总计 19 个错误类型，我设立了一个枚举类型 SemanticErrorType 来表示他们，并通过将对应的信息传入 SemanticError 函数来进行对应错误信息的打印。对于函数的参数，我同样用链表来表示，并对每个参数节点创建了一个类型为 FuncParameterType_的节点来存储相关的参数类型信息。

此外，在实现和运行的过程中也发现了实验一中一些由于大意和疏漏而导致的错误，也一并进行了排查和修复，实验二得到的文法处理程序在进行词法分析和语法分析时的准确性和鲁棒性也变得更强了。

编译该程序的方法为：

1. 将文件解压后进入到 Code 文件夹下；
2. 在终端中输入 make clean 命令并回车，防止有冗余残留的多余文件；
3. 在终端中输入 make 命令并回车，对程序进行编译；
4. 生成了最终的可执行文件。