**2025春《编译原理》实验二实验报告**

姓名：全俊康

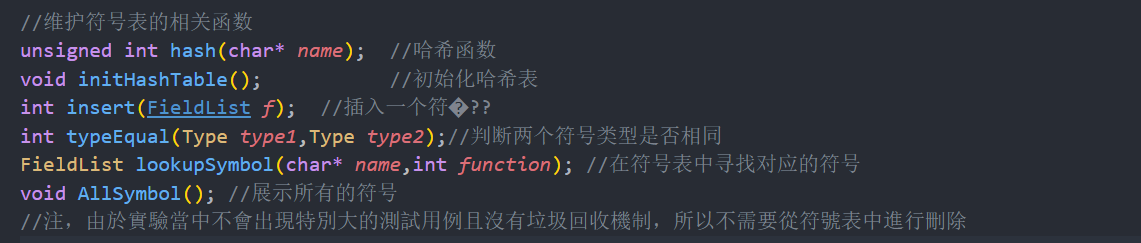
学号2022111553

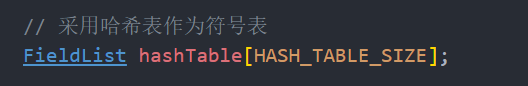
1. **主要功能和模块**

本次实验当中，在实验一的基础之上，完成了对语义的分析，主要通过semantic.h和semantic.c两个文件对语法树中的各个节点进行处理。

1. **符号表管理**

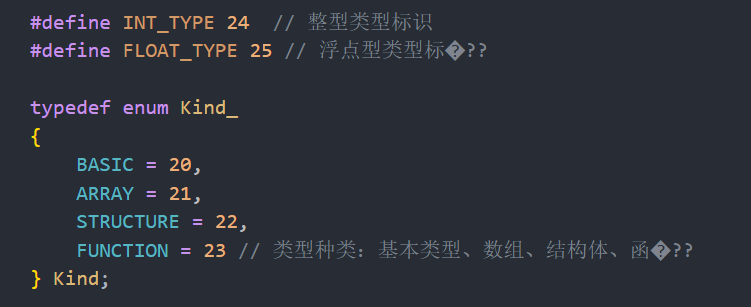
本程序中采用较为简洁的哈希表作为符号表，将实验指导书提供的哈希函数作为该哈希表的哈希函数，并针对不同的符号类型进行一定的偏移。采用线性探测法作为哈希碰撞的解决办法。实现了向符号表中插入，查询某一符号以及比对两个符号是否相同的函数，辅助检测出潜在的语义错误。





1. **符号定义与维护**

为了进行语义分析，我们需要将代码中出现的符号进行存储，这就需要为符号声明好一定的结构体。实验指导书已经给出了示例，即FieldList和Type，这里我将其进行了一点拓展，包括将符号类型抽出来作为一个枚举类等。

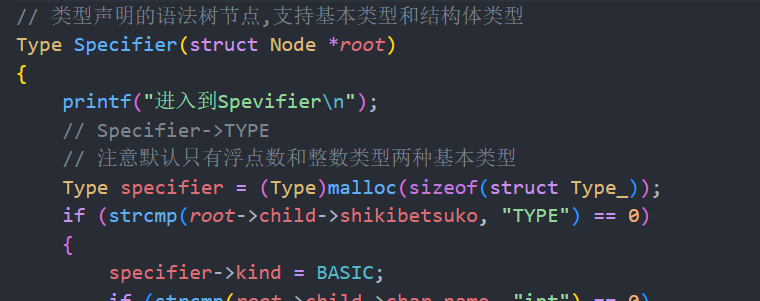


1. **语法节点处理**

语义分析的核心在于对语法节点进行处理并检测其中可能出现的语义错误，所以在此处我为C--文法中出现的非叶子节点的文法符号按照他们的文法都写了处理函数。整个程序的根节点为Program，之后紧接着ExtDefList，我们可以通过这个来遍历整个语法树，并在ExtDefList的处理函数中写好递归和调用的逻辑来确保每个语法符号都被分析到。例如下图的VarDec();

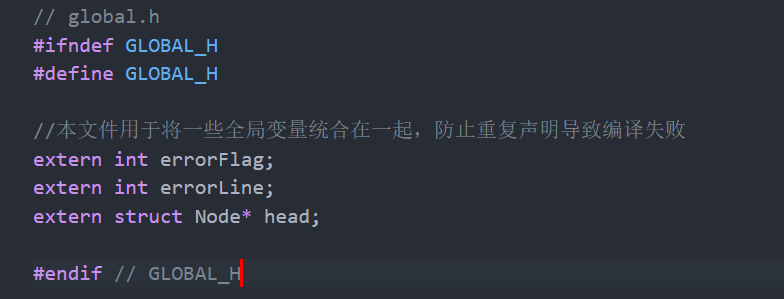
这里分析语法符号节点的时候是通过节点的识别符号（shikibetsuko），emptyFlag和不同文法产生式的长度来判断当前需要分析的是什么哪一个产生式。不同的函数根据语义返回不同的内容，例如VarDec()函数对应着程序当中的一些变量的声明，那么就应该返回一个FieldList。又比如Specifier对应着程序中的变量之前的类型，那么就应该返回Type





1. **全局变量管理**

相对于第一次实验，本次实验引入了更多的文件，我在写的时候经常遇到代码中变量或者标识符引用的问题。所以干脆将语法树相关的函数和flex/bison中用到的全局变量从第一次实验当中的syntax.y和lexical.l抽取出来，形成了global.h和node.h头文件，方便程序的引用和编译。



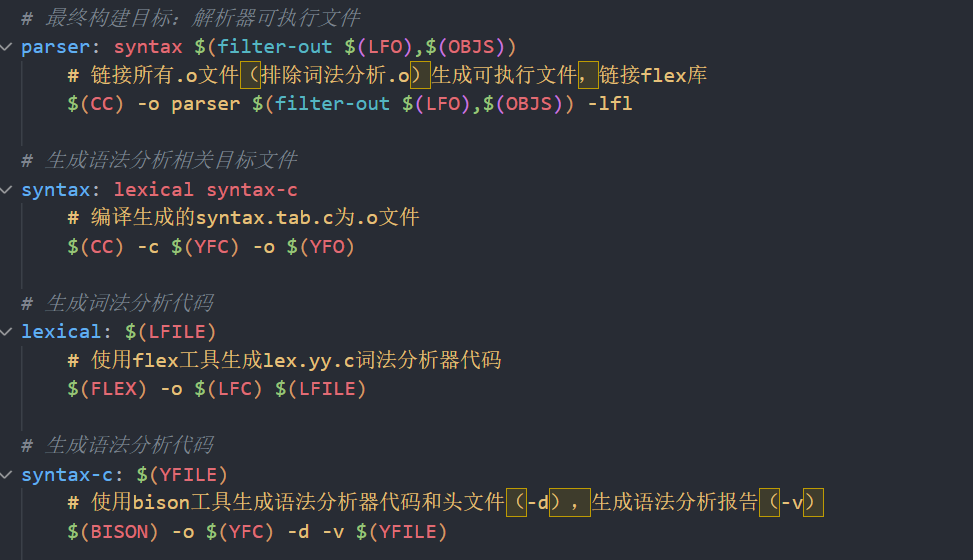
1. **编译过程**

使用了makefile文件进行编译，借助了一下AI大模型进行更快的编写。makefile文件比手动编译和用sh脚本都要更加方便一些，可读性也更高一些。

实验二的编译流程和实验一很相似，几乎不需要做出太多改动

主要编译的流程为

1. 先利用flex进行词法分析程序lexical.l的编译，生成lex.yy.c文件
2. 使用bison对语法分析的程序进行编译，主要生成syntax.tab.c和syntax.tab.h文件
3. 将上一步编译出的syntax.tab.c文件编译为syntax.tab.o文件
4. 将semantic.c文件也编译为semantic.o文件
5. 通过上述步骤将.c文件都编译为.o文件之后就可以进行连接，生成parser文件，即可读入其他程序文件进行编译



此外为了方便测试和文件管理，makefile文件中还规定了两个伪目标test和clean，前者会用生成好的parser文件对test文件夹下的测试用例进行编译，后者会清除掉编译过程中生成的其他文件，只保留parser文件以使得目录更加简洁。

Makefile文件写好之后，只需要进行（1）make （2）make test （3）make clean三种命令即可完成编译，比较方便。