实验报告

181850236 张子辰（基础班） 181850236@smail.nju.edu.cn

1. 程序运行和本地环境：

进入”/Lab/Code”目录，直接在终端运行 make 命令，即可编译生成 parser 文件，可以使用该文件对.cmm 文件进行分析。

我的本地运行环境为 Ubuntu amd64 18.04, flex 2.6.4, bison 3.0.4, gcc 7.5.0。

2. 程序功能

我的程序实现了本次实验的必做任务和所有选做任务。

本次实验中我的主要代码分成两个部分：hash.c/h 文件负责维护一个以哈希表为基础的符号表，semantic.c/h 文件负责分析语法树，利用符号表进行语义分析和报错。

为了便于进行语义分析，我还对 L1 中实现的语法树进行了修改，对每个节点加入了 prod_id 域，表示当前节点是选择哪一条产生式进行规约的。

实验中所有变量名、函数名和结构体名都存储在同一张符号表中。

为了区分结构体名与普通变量，我将结构体定义为一个带头的链表，其中链表头存储结构体名。同时，我让代表结构体名的 hash node 具有结构体的名字，而普通变量 hash node 名与结构体链表头的名字应该是不同的。

虽然一个函数可以进行声明和定义，但是我在符号表中仅保留一个，重复的声明不会被加入符号表，而遇到定义后会先删除原有的声明，再将定义插入。

为了实现错误类型 18 的报错，我还对函数单独创建了一个链表，以存储每个函数声明或定义出现的行数。在遍历完整棵语法树后，会遍历这个函数链表，检查是否存在只声明未定义的函数。

为了实现要求 2.2，我实现了 DepthStack[]数组来对不同深度的域进行管理，使用全局的 depth 来访问当前域。因为在插入符号表和 DepthStack[depth]时采取了相同的顺序，所以当从一个域中退出时，每次只需要从符号表对应链表的头部进行删除。

因为错误类型 15 和错误类型 3 是有一部分重复的，都表现为变量重复定义，只是错误类型 15 的变量是在结构体内定义的。为了区分这两种情况，我定义了 isStruct[]数组，来表示当前 depth 是否是在结构体中。

3. 程序亮点

① 我使用了上一级拔尖班同学维护的测试库进行了测试，可以通过 78/82 的测试样例，其中 m18.cmm、officialA-11.cmm、officialB-2.cmm 都是因为多报错误而未通过，但我认为我多报的错误是可以接受的，我可能产生多报错的部分情况如下：

a) 如果在算术运算中某个运算分量出错（包括变量未定义、域未定义、将函数名或结构体名加入运算等），则额外报一个错误类型 7，并返回某一个操作数的类型（优先返回左操作数类型）。
b) 如果赋值运算中出现某个表达式出错（同 a），则额外报一个错误类型 5，且优先返回赋值号左边表达式的类型。
c) 如果 if 或 while 语句中的表达式出错（同 a)，则额外报一个错误类型 7。
d) 如果 return 语句后的表达式出错，则额外报一个错误类型 8。
e) ……

另一个未通过的测试样例为 ZZ03.cmm，它内部定义了两族循环定义的结构体，我的程序在运算时会超时。我认为可以通过在定义结构体时检查所有已存在的结构体，将结构等价的结构体关联起来，从而加快运算过程。但可能会引入一些其他错误，我并没有实现该方法。