实验报告

151160055 吴宇昊 地理与海洋科学学院

程序如何被编译：

将 makefile 里的语句复制粘贴入 linux 终端即可完成编译与测试

程序功能：

符号表的格式使用实验指导的样例，相关声明与函数储存在 symbol.h 与 symbol.c 中

struct Type_结构体表示类型。kind 表示该类型的基本类型，0 表示基础类型 int|float，1 表示数组，2 表示结构体；在 union：u 中 basic=0 为 int 型，1 为 float 型；结构体 array 中 Type elem 表示数组下一维的类型，size 表示数组该维度的大小；FieldList structure 表示结构体的符号表

struct FieldList_结构体表示域，符号表的存储方式为散列表，使用实验指导提供的哈希函数。name 为域的名称，type 为域的类型，tail 为下一个域，hash 为当不同名的符号哈希值一样时采取的挂表措施所悬挂的指针

struct Functype_结构体表示函数表中的元素，存储方式同符号表。name 为函数的名称，type 为返回值的类型，param 为函数或结构体的参数链表，用于结构体，函数中的参数的存储搜索，hash 为哈希表的悬挂指针

void table()表的初始化

int insert_table(FieldList now)符号的插入，若该符号，数组名，结构体已存在则返回 1，反之进行符号的插入并返回 0

int insert_func(Functype now)函数的插入，同符号的插入

struct FieldList_* find_symbol(char *name)符号的查询，若已存在则返回该符号所对应的指针

struct Functype_* find_func(char *name)函数的查询，同符号的查询

语义分析的声明与函数储存在 semantic.c 与 semantic.h 中
语义分析的函数名均为语法中的非终结符，在 syntax.y 中在最终规约为 Program 时调用

void Program(struct sign *now)即开始语义分析

语义分析的函数参数列表的参数类型有符号表中的 **struct FieldList_，struct Type_**指针与词法分析语法分析中的 **struct sign** 指针

因涉及大量指针，链表操作，容易出现内存泄露，在语义分析中注释掉的奇奇怪怪的 printf(“”); 是用作于程序的调试，作为程序的标识符，了解程序错误的原因
在许多函数中所做的动作无非

1.根据语法中该非终结符下层的非终结符/终结符顺序，使用 **struct sign** 指针进行非终结符的移动，用 strcmp(“type of struct sign”,“name of symbol”)进行路径的选择，调用同名函数/类型的判断
2.进行符号表中相关结构的创建，主要集中在

struct FieldList_* VarDec(struct sign *now,Type type)变量结构体的创建，若有重名返回 NULL，若为数组的声明，则递归调用并创建新的类型结构构成数组类型的链表

struct Functype_* FunDec(struct sign *now)函数结构体的创建，重名返回 NULL

Type Specifier(struct sign *now)若为基本类型进行该类型的创建并返回，若为结构体则跳转到 **Type StructSpecifier(struct sign *now)**函数

Type StructSpecifier(struct sign *now)结构体类型的创建

Type Exp(struct sign *now)若规约为 int|float，则创建类型结构体，用于表达式中关系符左右两边类型约束的判断
在上述函数中创建了初始的结构体，然后才能在后续函数中进行指针的传递
剩余操作基本都是按部就班，根据语法中的非终结符/终结符进行操作，判断
类型的比较仅仅比较了 kind 与 basic，即基础类型的判断，对于数组与结构体的类型并没有进行判断

总结

非常抱歉延期了一天才提交，实验二的内容并不算难度很高，但是很的繁琐，对于链表，指针的操作容易出错，第一次使用 linux 的环境编写程序对于“段错误（吐核）”这种报错方式也是哭笑不得。不过从符号表的创立，部分成员的使用，到在语法分析中加入少量操作确定准确性，再到编写一个独立完整的语义分析系统，将错误一个一个的调试完毕，还是很有成就感的。实验有借鉴别人的整体思路和技巧，大部分为自己编写。