编译原理实验二实验报告

组号: 21	学号	姓名	联系方式
组长	161220096	欧阳鸿荣	895254752@qq.com
组员	161220097	戚赟	986300572@qq.com

一、实验目标和完成的功能

1.1 实验要求：

本次实验的任务是在词法分析和语法分析程序的基础上编写一个程序，从而正确、高效地实现语义分析的各种功能：对 C 源代码进行语义分析和类型检查，并打印分析结果。程序需要对输入文件进行语义分析并检查 17 种类型的错误（同时还有选做要求）。

1.2 完成功能：

我们完成了**所有的基本功能**，同时**实现额外要求 2.2**：嵌套作用域下的变量定义。

二、实验实现方式

2.0 文件结构:

比实验一增加的文件:

“semantic.h”：语义分析和类型检查头文件，主要包含各种类型的表示与定义以及支持嵌套作用域的基于十字链表和 open hashing 散列表的符号表，还有各种处理函数。

“semantic.c”：各种处理函数的实现，以及一些辅助调试的函数。

2.1 实验设计

本次实验将语义分析单独作为一块内容，基于语法树进行分析。对于没有错误的输入文件，首先执行实验一的内容，进行语义分析，同时成功构建语法树。之后我们基于语法树进行分析。从语法树的根开始，以深度优先的遍历方式自上而下对语法树进行分析。对于语法树对应的各个有意义的产生式，都对其设置 Handle 处理函数，通过产生式的语义，对语法书逐层处理，并在 Handle 函数中处理符号表的插入、查询等操作。

2.2 数据结构设计

由于需要实现选做 2.2 的嵌套作用域内容，因此本次实验的符号表采用基于十字链表和 open hashing 散列表的符号表。首先定义类型 Type 表示 C 语言中的类型，同时有对于结构体和多维数组采用以 FieldList_为节点的链表表示:

```
1.  enum { INT_TYPE, FLOAT_TYPE };
2.  struct Type_ {
3.      enum { BASIC, ARRAY, STRUCTURE } kind;
4.      union{
5.          int basic;  // 基本类型
```

```
        struct { Type elem; int size; } array;  // 数组类型信息
        FieldList structure;                // 结构体类型信息是一个链表
    } u;
};
```

有了自定义的类型系统后，便可以自定义符号类型，包含的信息有符号名，符号所在行号，符号类型等，并用 union 类型分别表示各个类型的信息。right 表示在散列表指向同一个哈希值的符号，down 指向同一作用域的符号。

```
struct SymbolElem_ {
    char name[30];
    int lineNo;
    enum { VAR_ELEMENT, FUNCTION, STRUCTURE_ELEMENT } kind; //三种类型定义
    union {
        Type var;   //存储 VAR 和 STRUCTURE
        struct {
            Type retType;
            FieldList varList;
            int complete;   //complete 代表函数是否被定义完整
        } func;     //存储函数
    } u;
    SymbolElem right;    //open hashing
    SymbolElem down;     //local GrammarTree chain
};
```

定义符号表 symbol_HashTable 和作用域的栈 symbol_Stack，并用一个变量 Top_of_stack 记录栈顶的位置。

```
SymbolElem symbol_HashTable[MAX_HASHNUM]; //define hash table
SymbolElem symbol_Stack[MAX_STACKNUM]; //define symelem stack
int Top_of_stack = -1;  //the top of stack
```

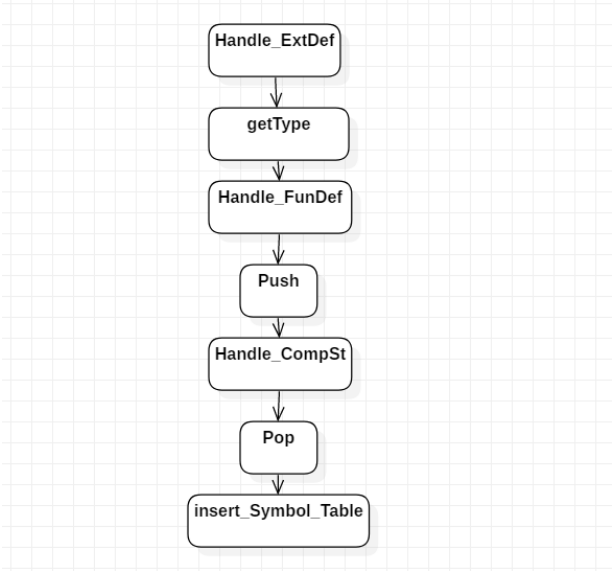
2.3 必做部分完成

错误分析：在必做部分共有 17 种错误类型，其中错误类型 1-4 本质上都是完成对符号表的检索，在定义时要求符号表内没有同名符号。若无则新建该符号，并将其加入符号表。在调用时要求符号表内已有该符号，且类型要一致。错误类型 5-9 都是对符号类型的检查，对于不同的运算符的不同类型要求，其中值得注意的是判断表达式是否有左值。错误类型 10-14 是对表达式中数组访问操作符 [...]，函数调用操作符 (...) 和结构体访问操作符. 的检查，此部分着重在 Handle_Exp()中进行处理。错误类型 14-17 都是对结构体的检查。定义结构体时，从符号表中查找该结构体名称，判断是否重复定义，然后再从符号表中查找该名称，判断是否和已定义的变量名字重名。在结构体内部，域名不能重复定义，也不能对域初始化。访问结构体时，从类型列表中查找该结构体名称，若未找到则报错。

处理流程：从语法树的根进行遍历，以 ExtDef 作为初始分析的入口。每个 ExtDef 表示一个全局变量、结构体或函数的定义，通过判断选择处理的分支。ExtDef 具体的处理分为三

个部分，第一个部分是"ExtDef → Specifier ExtDecList SEMI"，这部分主要是处理变量的声明，处理时利用 getType(Specifier)函数来获得变量的类型，再处理 ExtDecList 获得变量名等，最后进行查表和插入表。第二个部分是 "ExtDef → Specifier SEMI"，这部分特殊处理结构体的定义，利用 getType()函数处理 Specifier 的定义，从而得到结构体变量。最后"ExtDef → Specifier FunDec CompSt "部分进行函数的处理。由于错误是返回值类型和函数类型不匹配的问题，所以在处理的时候，先处理 Specifier 得到一个 Type，再传参处理 CompSt 等才能比较方便的处理某些错误。

函数定义处理样例图：



2.4 选做部分完成

选作部分的主要内容是变量定义的嵌套作用域。以函数分析为例，在 HandExtDef()处理函数中，若产生式表示函数的定义，首先对于 Specifier 通过 getType 得到返回值的类型。接着通过 Handle_FunDec()函数处理函数名以及形参定义，并得到一个函数符号 Func。若有完整的函数定义，便进入了嵌套作用域名，此时让作用域栈指针加一，使用 Handle_CompSt()函数处理，并在处理完成后，用 Clear_TopOf_Stack()退栈进行现场恢复，并将符号 Func 插入符号表中。同理，对于 while 和 if 语句等，也是在处理语句块 CompSt 前让作用域栈指针加一，处理后用 Clear_TopOf_Stack()退栈进行现场恢复。通过这种设计，便可以很好地支持变量定义的嵌套作用域。

除了上述工作，我们的代码还具有良好的测试打印输出和比较多的额外测试用例来保证实验的相对正确性。

三、实验编译方式

3.1 实验的编译方式如下:

① 进入 **Lab2/Code** 目录下

② 执行命令 **make clean**，再执行 **make parser** 命令（先执行 make clean 防止先前文件定义冲突）

③ 生成目标程序 **parser**，执行命令 "**./parser test.cmm**"来启动 parser 对于 test.cmm 的分析。或者改变 makefile 中 make test 的.cmm 文件位置然后执行'make test'操作即可。其中提供测试文件放在 **Lab2/Test** 目录下。