Lab2 实验报告

191870147 屈力

实现功能

完成了所有内容（必做+选做）

1、使用哈希表实现了符号表，哈希表的节点的结构如下

```
struct SymbolNode_{
  FieldList val; //符号值
  unsigned int hash_val;
  SymbolNode next_node; //下一节点，用于每个槽位的链表
  SymbolNode prev_node; //上一节点，用于每个槽位的链表
  SymbolNode tail_node; //属于同一作用域的下一个节点
};
```

在哈希表的每个槽位对应一个链表，相同哈希值的节点会被映射到相同的链表上。该结构体中，$hash_val$和$prev_node$成员可以省略，但保留可以节省哈希表相关操作的查找时间。

2、 通过对语法树的遍历实现了语义分析和类型检查。对于每个非终结符，写了一个对应的处理函数。对于每个非叶子节点，通过他子节点的情况决定对应的产生式，按序调用对应的处理函数。对于不同的非终结符，它的处理方式、返回值、参数也不尽相同，具体细节省略。总体来说，在函数互相调用的过程中，一旦遇到变量或函数名，可以通过产生式判断它是定义还是使用，抑或声明（函数专用），以及它的类型（如果是结构体，还要包括它的成员变量；如果是函数名，还要包括它的参数），然后通过查找符号表（如果是空且当前要定义变量，则向符号表插入节点）实现语义分析与类型检查。我在symbol_table.c里定义了几个专门负责检查的函数：

```
int check_function_define(FieldList old, FieldList new);//检查函数的声明和定义是否一致
FieldList find_member_in_structure(FieldList f, char *var_name);//查找结构体中的成员
int check_type(Type a, Type b);//检查两个变量的类型是否一致
int check_args(FieldList a, FieldList b);//检查两个函数的参数是否一致
```

3、允许函数声明的存在，只需新增ExtDef→Specifier FunDec SEMI产生式，并在Type结构体中新增一个has_def成员变量，当程序在语法树中扫描到该产生式时，将函数作为节点插入到符号表中，并标记has_def为0；当下次遇到函数声明时，检查两个声明是否一致；当下次遇到函数定义时，检查两个声明是否一致，若一致，则修改has_def为1。除此之外，需要维护一个函数定义\声明链表。当整棵语法树遍历完后，需遍历链表，查询是否存在声明但未定义的函数。

4、实现了作用域。除了结构体名字（Tag），都需要实现生命周期。我实现了两个函数:

```
ScopeList create_scope();
void delete_scope();
```

分别实现创建作用域，删除作用域的功能。在全局开始、遇到CompSt、遇到LP时调用create_scope；在全局结束、处理完CompSt、遇到RP时调用delete_scope。Scope的结构如下：

```
struct ScopeList_{
  SymbolNode head;//当前作用域第一个槽位
  SymbolNode tail;//当前作用域最后一个槽位
  ScopeList next;//下一作用域
};
```

Scope用栈存储，每创建一个Scope，将新建Scope压栈；删除时弹出。因此有一个很好的特性：当前总是处于栈顶指向的作用域。因此很容易判断符号表查到的节点是否处于当前作用域中，如果不处于，局部变量可以无视已创建的那个，而是再次插入符号表，实现了要求2.2的功能。

5、实现了结构体的结构等价，实现方法和讲义相同，不多赘述。

编译方法

在当前目录下直接make（我删除了Makefile的-ly选项，因为这个选项一般没有影响，但是加上后我这里(Ubuntu 20.04)无法编译）。
在终端输入./parser ../Test/filename可以测试某一文件，或修改Makefile中的test目标然后直接输入make test以测试某一文件。

实现亮点

1、多使用assert断言。在合适的位置编写恰当的断言可以有效地及早发现bug，尤其是在实验二这种对细节处理要求很高，容易出错的地方。比如，我在大多数产生式处理函数开始时都会assert(right_num==num)，（num为子节点数目，right_num为产生式右部符号数目），由于实验一要求语法树中不存在空的非终结符和终结符，这一断言在很多地方并不正确。我在调试时遇到该处的很多assert fail，帮助我很快认识到编写代码时发生的一些逻辑错误。另外，在链表操作时assert也很有效。