编译原理二实验报告

211220127顾嘉宇

实验功能

在本次实验中，我的任务为**要求2.2**，在实现符号表存储检索和类型检查的基础上，修改前面的C语言假设4，使其变为“变量的定义受可嵌套作用域的影响， 外层语句块中定义的变量可在内层语句块中重复定义，内层语句块中定义的变量到了外层语句块中就会消亡，不同函数体内 定义的局部变量可以相互重名”。

在词法分析和语法分析均正确之后，采用自顶向下的方式进行语义分析，进行符号信息的添加维护和类型检查。

符号表

对于符号表，我采用了哈希表的实现方式，并采用开散列的方式处理冲突。

符号类型

```
struct Type_{
    enum{BASIC = 0,ARRAY = 1,STRUCTURE = 2,STRUCTTAG = 3,FUNCTION = 4}kind;
    union{
        int basic;//int(1),float(2)
        struct{
            Type elem; int size;
        }array;
        Structure structure;
        Function function;
    }u;
};
```

- `kind`为其基本类型，分为五类，其中`structure`和`structtag`分别为结构体变量和结构体定义名
- `basic`为基本类型定义，分为`int(1)`和`float(2)`
- `array`为数组类型，其中用`type`链表将每一层串联起来，`size`即为每一层的大小
- `structure`为结构体类型，根据`kind`将变量和定义名区分开
- `function`为函数类型

符号域

```
struct FieldList_ {
    char* name;      // 域的名字
    Type type;       // 域的类型
    FieldList tail;  // 下一个域
};
```

符号域包含符号的类型和符号的名字，结构体域以及参数列表都采用这种链表形式串起来。

函数和结构体定义

```
struct Structure_{
    char *name;
    FieldList domain;//结构体中的域
};

struct Function_{
    char *name;//函数名字
    int line;//
    int para_num;
    Type type;
    FieldList param;//函数的参数列表
};
```

哈希节点和哈希表链表

```
struct HashNode_ {
    char* name;//节点名字
    Type type;//节点类型
    HashNode link;//链接该节点的下一个节点
};

struct HashTableLink_{
    HashNode hashtable[HashSize+1];//该哈希链表中的哈希表
    HashTableLink next;//在创建新哈希链表时指向下一个哈希链表
    HashTableLink before;//在删除当前哈希链表时指向前一个哈希链表
};
```

对于**选做任务2.2**，我使用的是`Functional Style`，即语义分析器读到`CompSt`时在栈顶新建一个哈希表，并在执行结束时删除栈顶的哈希表，需要注意的是，由于在读取完函数的参数列表后，才会将函数类型的符号插入到符号表中，所以插入时需要注意将其插至栈顶层数下一层的符号表中，这样在删除栈顶符号表时不会讲该函数符号给删除。

编译&测试方式

进入`Code`文件夹所在路径，执行`make`命令，即可获得`parser`可执行文件。 执行`./parser test.cmm`命令，即可对一个待测试的源文件进行词法、语法、语义的分析。

实验感悟

本次实验是实现编译器的语义分析部分，涉及符号表的存储检索以及类型检查，该实验难度思路也十分清晰，即顺着语法树的展开逐步对每个语法单元进行相应的处理。但符号表的设计、符号插入的时机、类型检查的具体逻辑顺序，都需要认真查看。并且最关键的是，实验涉及较多的指针赋值操作，导致经常容易发生`Segmentation fault`的类型错误，因此编写各种`debug`函数，在合适的位置插入断言和熟练使用gdb调试工具便十分重要。