实验报告

171240501 匡舒磊

kuangsl@foxmail.com 基础班

已实现内容

完成了所有必做内容与选做内容,可以尽可能多的判断语义错误。

代码框架

`lexical.l` : 词法内容识别

`syntax.y` : 语法内容分析，并建立相应的语法树节点

`common.h` : 函数定义与相关结构体定义

`ast.c` : 语法树的相关函数，包括添加节点打印语法树等

`main.c`: 运行分析

`envStack.c` 维护多层作用域等操作

`openhash.c` 与哈希表操作相关，包括插入删除查询等操作

`semantic.c` 语法分析过程

`error.c` 报错信息输出

如何编译

使用助教提供的Makefile进行编译生成parser

实验环境

GNU Linux Realease: Ubuntu 18.04.4 LTS

GCC version 7.5.0

GNU Flex 2.6.4

GNU Bison 3.0.4

实现的重点细节

1. 符号表的维护：实验的符号表采用了实验报告中的方式，基于**十字链表**与**open hashing 散列表**而成，值得说明的是在实验中考虑到允许局部变量的存在，但是结构体的名字是全局作用的，所以在退出一个作用域的时候，其中的结构体定义不会消失。
2. 数据结构的采用选择了讲义中的数据结构，同时定义了函数对应的结构体与所有符号共同组成的结构体（common.h）,其中符号表中的数据结构定义如下：

```
struct Symbol_
{
    char* name;
    union
    {
        Type type;
        Func func;
    }t;
    Symbol hashNext;
    Symbol stackNext;
    int depth;
    int isfunc;
    int isdef;
};
```

其中hashNext指向拥有相同hash值的下一个符号，stactkNext指向在同一个作用域中的下一个符号，depth的作用在于判断变量重名时是否判断两个符号是否在同一作用域（利用空间开销避免遍历一遍链表），isfunc判断是否为函数定义，isdef判断是否为结构体的定义。

3. 语义分析过程如下：先对符号表进行创建与初始化，然后从语法树的根节点开始，每类产生式（指左侧相同）都对应一个函数，然后分别进行分析。最后判断是否有未定义函数。

实验中遇到的困难/不足

1. 原本想实现申请内存后的free，但是实在bug太多了（/(ＴoＴ)/~~），最后决定放弃free只申请不释放，在超强测试的大输入时或许可能会出现问题。
2. 在结构体和函数的局部变量定义中函数理论上可以做到复用统一成一个接口，但是因为写的顺序的原因加上重构比较复杂就放弃了，导致代码略微冗长。