编译原理课程Lab2

班级	队伍	组长	组员	任务号	联系邮箱
普通班	2人队	181870290 周心瑜	181860127 姚逸斐	16	181870290@smail.nju.edu.cn

实现功能

本次实验在实验一词法分析与语法分析的基础上，完成了语义分析和类型检查，并打印分析结果。完成了必做内容和选做内容2.3。

本次实验的主要实现都在`semantics.h`与`semantics.c`中。功能实现的细节可以进一步细分为如下：

符号表

本次实验采用哈希表来组织符号表，用拉链法来解决冲突。具体数据结构为：创建一个大小为0x3fff的数组，数组中的元素为HashNode，其主要属性是kind（判断是基本类型int/float，还是结构体，还是函数），和一个联合体表示变量和函数（结构体当作变量处理）：若节点是函数，则含有返回值，参数和名字；若节点是变量或结构体，则含有名字和type，其中type含有一个联合体，依据具体类型的不同（基本类型/数组/结构体）含有不同的成员（基本类型/数组元素类型和大小/域）。以及一个next指针连接下一个相同哈希值的元素。

```
typedef struct HashNode{
    HashNodeType kind;
    union {
        Variable variable;
        Function function;
    }u;
    struct HashNode* next;
}HashNode;
```

相关函数有：

```
extern void HashListInitial();
extern unsigned int Hash_pjw(char* name);           //Hash Function
extern bool HashListAdd(HashNodeType node_type, void* u, int current_line);

//insert successfully return true; else false
extern void* HashListFind(HashNodeType node_type, char* name);
```

其中填表部分，根据具体类型创建不同的HashNode，计算哈希值，若无冲突则直接插入，若有冲突则插入到链表末尾，在这过程中检查是否有重名，若有报错类型3，否则顺利插入。

至于查找部分，根据给定的名称计算哈希值，在对应的桶内查找，若找到则返回对用的变量/函数/结构体，若未找到则返回空指针。

语义分析

大体而言，基于实验一的语法树进行遍历并填表供分析。关于信息的传递，是通过SyntaxTree的结点，每一层check使用的是哪一个上下文无关语言的句柄展开，然后调用，把综合属性作为返回值在调用结束后整合，继承属性在调用函数的参数中代入。主要函数如下：

```
//************** Specifiers ****************************
extern Type Specifier(TreeNode* root);
extern Type structSpecifier(TreeNode* root);
extern char* OptTag(TreeNode* root);
extern char* Tag(TreeNode* root);
//************** Declarators ****************************
extern Variable VarDec(TreeNode* root, Type t, bool in_struct_field);
extern Function FunDec(TreeNode* root, Type t);
extern FieldList VarList(TreeNode* root, int* count);
extern Variable ParamDec(TreeNode* root);
//************** Statements ****************************
extern void CompSt(TreeNode* root, Function f);
extern void StmtList(TreeNode* root, Function f);
extern void Stmt(TreeNode* root, Function f);
//************** Local Definitions ************************
extern FieldList DefList(TreeNode* root, bool in_struct_field);
extern FieldList Def(TreeNode* root, bool in_struct_field);
extern FieldList DecList(TreeNode* root, Type t, bool in_struct_field);
extern FieldList Dec(TreeNode* root, Type t, bool in_struct_field);
//************** Expressions ****************************
extern Type Exp(TreeNode* root);
extern void Args(TreeNode* root, FieldList f, char* function_name);
```

另外check相关的函数如下：

```
//============== Checking Conditions ======================
//error 5/7/8
extern bool isSameType(Type a, Type b);
//error 6
extern bool isLeftValueExp(TreeNode* root);
//error 7
extern bool isLogicExp(Type a);
extern bool isArithExp(Type a);
//error 15
extern bool SharingSameName(FieldList f, int current_line);
//2.3
extern bool isSameField(FieldList a, FieldList b);
```

编译方法

实验编写环境为：

Ubuntu	GCC	Flex	Bison
16.04	5.4.0	2.6.0	3.0.4

使用makefile进行编译，在命令行输入`make`编译；输入`make test`对Test目录中的文件进行测试，也可以使用`./parser ../Test/test.cmm`对某一指定文件进行测试。

实验感悟

- 实验中遇到的问题与解决方法
  - 需要注意实验一代码到实验二代码的复用性。如：
    - 基于lab1生成的语法树将生成空串的语法单元省略，而之后lab2还原语法分析需这些节点。故将lab1时采取了不生成的做法在本次实验中进行更改。
    - 所有错误的行号问题：在lexical token也需要记录行号。
    - 需要增添两个接口，用于返回语法树节点的子节点信息。
  - 需要注意实现代码时的错误避免。如：
    - struct type参数错误
    - 由于ParamDec拼写错误造成定义形参不会进行重复判断
    - StructSpecifier大小写错误以至于与实验一数据无法匹配
  - 需要注意编译样例中的边界条件。如：
    - struct定义成员域为空时仍需要建立节点，否则这个结构类型将不会加入符号表。（测试m6.cmm时发现的错误）
    - test12中对于[]中的浮点数，起初的判断未包含这一情况
  - 需要注意程序的鲁棒性。
    - 在使用指针前务必判断其是否指向特定对象。如：在判断逻辑/算数运算时，注意指针是否为空，否则可能出现段错误。
    - 在检测出产生部分语义错误后，不能单一的将该节点的综合属性设置为NULL，而应该进行错误恢复。如：在Exp函数中，Exp->Exp RELOP EXP类型不同时仍需要返回int类型以做错误恢复。否则在IF与WHILE语句中判断是否为逻辑运算类型时将会出现错误。
- 在本次实验中，运用到了大量指针，有内存泄漏的隐患，这个也许可以进一步改进。
- 感谢https://github.com/massimodong/compilers-tests提供的测试数据