编译原理第二次实验报告

小组信息	181860087	唐业
	181860102	王印可
任务信息	编号3	必做内容 + 选做2.3（结构等价）

1. 实现的功能

1.1. 实现功能

- 对输入文件进行语义分析并检查所有的错误类型（必做内容）；
- 将结构体间的类型等价机制由名等价改为结构等价（选做2.3）。

1.2. 数据结构

1. `SymbolTuple`（表示符号表其中一个符号的结构体）

```
struct SymbolTuple
{
    char name[32];
    Type type;
    Symbol hashLink;
};
```

符号表采用哈希表形式，处理冲突的方式是close addressing，即如果出现冲突，就存放在相应哈希值数组元素的hashLink处。

2. `Type_`（表示类型的结构体）

```
typedef struct Type_ *Type;
typedef struct FieldList_ *FieldList;
typedef struct FuncList_ *FuncList;
struct Type_
{
    enum {  BASIC, ARRAY, STRUCTURE, FUNCTION, STRUCTVAR, ERROR} kind;
    union
    {
        int basic;
        struct { Type elem; int size;} array;
        FieldList structure;
        FuncList function;
        int errorCode;
    } u;
};
```

BASIC表示基本类型(INT/FLOAT)，ARRAY表示数组，STRUCTURE表示结构定义，FUNCTION表示函数，STRUCTVAR表示结构变量，ERROR表示错误。

结构体域内和函数参数信息使用链表存储，`FieldList structure`和`FuncList function`分别为两个链表的表头，其中链表的一个节点存储了变量的名字`char name[32]`、类型`Type type`、尾指针`tail`。

3. `StructSymbolTuple`（表示结构体域内的节点）

```
struct StructSymbolTuple
{
    char name[32];
    StructSymbol link;
};
```

之所以使用这个数据结构主要是是为了处理结构体域内的重名情况，判重的方法即使用哈希表。

1.3. 实现方法

1.3.1. 代码框架

为每个非终结符分别定义一个函数，即给语法树上的每个非叶节点定义一个函数，在每个函数中实现相应的功能，每个函数可以从父节点或子节点接收信息，同时向父节点和子节点传递信息。举个例子，文法定义中有这样一条`ExtDefList-> ExtDef ExtDefList`，那么我们定义以下两个函数：

```
void ExtDefList(node root);
void ExtDef(node root);
```

当然实现的时候没有为所有的非终结符定义函数，因为有些非终结符的功能实现比较简单，合并即可。

1.3.2. 错误类型检查

整体的语义分析主要分为两部分——**定义**和**语句**（归约到StmtList），

定义部分包括错误类型：3，4，5，15，16，17，

语句部分包括错误类型：1，2，5，6，7，8，9，10，11，12，13，14。

- 出现在定义部分的错误，处理逻辑是在插入符号表之前。先调用`findTuple`来检查，如果出现重复则报重复定义错误。需要注意的是，对于结构体域内的变量不需要对其进行符号表的查重，但域内重名还是需要检查，这里的实现方法是使用**1.2.3**中定义的哈希表来处理重名。当遇到定义中对变量进行赋值的时候，需要调用**1.3.3**中的`isTypeEqual`函数来判断赋值号两边的类型是否相等。
- 出现在语句部分的错误，处理逻辑在`Exp`函数中。先调用`findTuple`来检查，如果没有找到则报未定义错误，否则根据查到变量的类型来判断使用方法的正确，比如对非数组类型的变量使用'[]'，对非结构体的变量使用'.'。需要注意的是，如果一个语句中的某个Exp的的类型需要一个变量，但是根据符号表查到的结果是函数或者结构定义，此时即使查到了这个这个符号，但是因为没有正确使用这个函数或结构定义符号，会报未定义的变量，比如：`int a{} ..... a = 1;`，`a`是一个函数，但是显然没有定义一个a的变量。
- 对于错误类型6，赋值号左边出现一个只有右值的表达式，实现方法是修改了语法树的节点结构，增加一个`flag`域，初始化为0，当Exp遇到ID、Exp LB Exp RB、Exp DOT ID时，将根节点的flag设置为1，在赋值语句中判断左边节点的flag是否为1来表示这个节点的表达式是否为左值。
- 对于**1.2.1**中的ERROR类型，多出这个类型的定义主要是因为如果在下层的语句中有错误，那么直接返回一个NULL指针显然对上层语句的处理是不友好的，因为还需要判断下层返回的类型的是否为空指针，所以设计了这一个错误类型来告知上层语句哪里出现了错误，但在实现的过程中发现`errorCode`没有什么很大的作用。
- 当函数体的形参或结构体中的成员发生了语义错误时，我们不会将它们加入我们维护的`SymbolTable`与`StructSymbolTuple`，但与此同时，为了减少成员语义错误所带来的一些不必

要的连锁错误（例如后续判断等价等问题），我们并不会完全抛弃该函数或结构体的创建，而是选择继续解析后续的成员来尽力恢复出一个完整的函数或是结构体。

1.3.3. 选做部分结构等价

判断两个结构体是否结构等价的时候，步骤是查符号表获取Type类型信息，然后将这两个Type信息比较来判断是否相等，这边定义一个函数`int isTypeEqual(Type t1, Type t2)`来判断两个Type是否相等，实现的方法是分Type的kind来讨论，结构变量或结构体情况下，递归调用`isTypeEqual`来判断链表的每个节点的Type是否相等。

2. 编译运行

- 如何编译?
    1. 进入目录`Compiler/Lab/Code`
    2. 在控制台运行指令`make parser`
- 如何运行?
    - 命令行输入`make test`或`./parser  xxx.cmm`(`xxx.cmm`表示自己编写的测试输入文件)

以上便为实验二报告的全部内容