实验二：语义分析

2019.11.5

1 小组信息

组号	学号	姓名	邮箱
1	161220022	陈哲霏	novelist.chan@gmail.com
	161220024	成威	weicheng.nju@qq.com

2 实验说明

2.1 实验要求

2.1.1 必做

- 对C--语言做出语义规约，满足7个先决条件假设来表示当前语言的各项特性
- 对输入文件进行语义分析，检查17中基本类型语义错误并正确输出提示信息

2.1.2 选做

- 选做1：修改语言假设3，使得函数在定义之外还可进行声明，并增加错误类型18、19的判定

2.2 编译运行

- 在Lab/Code目录下执行make指令完成编译
- 执行make test完成对Lab/Test下所有测试文件的语义分析

3 实验内容

3.1 基于散列表的符号表定义

- 共分7个结构体定义，分别对应语言中不同的类型和散列表节点定义

```
typedef struct Type_* Type; //变量类型
typedef struct FieldList_* FieldList; //结构体或参数列表
typedef struct DecLine_* DecLine; //函数声明行号
typedef struct Func_* Func; //函数类型
typedef struct Info_* Info; //区分重名类型
typedef struct HashNode_* HashNode; //单个哈希节点
typedef struct TableNode_* TableNode; //哈希表单元
```

- 冲突时采用开散列、闭地址的方式存储
- 在讲义提供的代码基础上的扩展部分：
  - 定义了函数声明行号DecLine，用以在语义分析过后检查是否有声明但未定义的函数
  - 定义了Info结构体以链表形式存储重名定义的节点，如符号i被声明为变量之后又被声明为函数，此时识别为Error Type 4，之后的i将具有变量和函数的双重身份，可以被视为函数进行调用

3.2 语义分析函数

- 共设计了27个函数参与语义分析主体，通过semanticAnalysis进入，开始整体语义分析

```
void semanticAnalysis();    // entrance of semantic analysis
void Program(TreeNode *node);
void ExtDefList(TreeNode *node);
void ExtDef(TreeNode *node);
void ExtDecList(TreeNode *node, Type type);
// ......
```

- 为满足假设1、2的要求，设计了ArithmeticCheck和LogicCheck来判断算术和逻辑运算是否符合要求

```
// 算术运算符
Type ArithmeticCheck(Type left, Type right, int line);
// 逻辑运算符
Type LogicCheck(Type left, Type right, int line);
```

3.3 额外功能

- 修改syntax.y文件，加入函数声明的语法

```
ExtDefList : ExtDef ExtDefList
  | FunDeclaration ExtDefList
  | /* empty */
  ;
FunDeclaration : Specifier FunDec SEMI
  ;
```

- 在Func定义中加入ifDef和ifReal属性，判别当前函数是未声明、声明未定义还是声明且定义状态
- DecLine帮助定位声明未定义的函数

```
bool ifDef; // 是否已被声明
bool ifReal; // 是否已被实现
DecLine decLines; // 存储声明行号
```

4 实验总结

- 本次实验的最大难点在于数据结构的设计，为了区分实验中纷繁复杂的错误情况，我们设计了多个辅助结构体帮助判别语义错误
- 在我们书写的语义分析函数中，函数的运行顺序映射到了语法树的层级结构，这一理解在debug时给到了很大帮助
- 为了实现一些功能，我们设计了多个check和compare函数，用来检查当前行为的逻辑正确性和比较对应类型(函数、结构体、参数列表)是否一致
- 为了实现Exp中的错误行号打印，在Lab1的词法分析中对所有终结符与非终结符都加上了行号的标记，算是对Lab1的一个补充
- 在完成编译器的同时收获了快乐