编译原理第二次实验

实验报告

171860691 王辛有 171860005 宋斯涵

一、 实现功能

1） 进行语义分析并识别出所有的错误类型（必做部分）。
2） 将结构体名等价改进为结构等价（选做 **2.3**）。

二、 重要的数据结构说明

最重要的数据结构：表示类型的结构

```
struct Type_
{
    enum { BASIC, ARRAY, STRUCTURE, FUNCTION, ERROR } kind;
    union
    {
        //struct A x;//其中x是UserDef,A是Structure
        // 基本类型
        int basic;
        // 数组类型信息包括元素类型与数组大小构成
        struct { Type elem; int size; } array;
        // 结构体类型信息是一个链表
        FieldList structure;
        //函数名
        FuncList myfunc;
        //错误类型
        int error_type;
    }u;
};
```

该数据结构可以表示出所有可能出现的结构。

其中 BASIC 表示基本类型，ARRAY 表示数组，STRUCTURE 表示结构体，FUNCTION 表示函数，ERROR 表示错误（下文会提到 ERROR 的用法）。

结构体和函数的类型信息分别用 FieldList 和 FuncList 来表示（具体定义参见 SemanticAnalysis.h）：

符号表的数据结构

数据结构：

```
struct HashNode_
{
    char name[32];//节点名称
    Type val;//节点类型
    HashNode next;//哈希值相同的下一个节点
};
```

符号表采用散列表形式构建，其中处理冲突的方式是 close addressing。（如果散列表出现冲突，则可以在相应数组元素下面挂一个链表）

三、 实现方法

1. 代码整体框架

为每个非终结符都定义了一个函数，即：为语法树的每个节点都定义了一个函数，在每个函数中实现相应的功能，每个函数可以从父节点和子节点中接收信息，并且向父节点和子节点传递信息（通过参数和返回值）。

示例：

```
Type Specifier(Tree *root);
Type StructSpecifier(Tree *root);
```

参数为产生式左端的节点，函数会返回一个类型的信息。

2. 错误类型检查

整体代码分为两大部分：定义部分和语句部分（指会归约到 StmtList 的语句）。定义部分包括全局变量、函数（仅包含返回类型，函数名，形参的定义）、结构体、局部变量的定义，语句部分则是函数体内除去定义部分的相关语句。

错误类型 3,4,15,16,17 会出现在定义部分，

错误类型 1,2,5~14 会出现在语句部分。

- 出现在定义部分的错误，则会在插入符号表的时候进行解决。先调用 check 函数检查，如果出现重复则报重定义。在结构体的定义中，会对每个域名进行检查，如果出现域名重复同样会报错。
- 出现在语句部分的错误，则会在 Exp 函数中进行解决。本代码针对以 Exp 为左端的所有产生式划分了 13 个函数（具体定义参加 SemanticAnalysis.h），起到了良好的功能划分作用，在出错时可以很方便地定位。在检查每个语句时，会先去符号表中查找出现在该语句中所有变量的信息，如果没有查到则会报未定义的错误，如果查找到了，但是使用方法不对，比如对非结构体类型的变量使用"."操作符，同样会报相应类型的错误。
- 关于上文中 Type 类型里 ERROR 的说明：每个 Exp 相关函数都会返回一个类型。如果在函数里发现了错误（例如，操作符类型不匹配，类型未定义等），则会返回 ERROR 类型，来告知上层语句这里出现了错误。

- 附加说明：对于有些在 C—语言假设中出现，但是并未考察的错误，为了保险起见，我们也输出了错误信息，错误类型统一为 Error 18，例如：if 和 while 条件判断语句中的数值不是不是 int 型变量。

3. 选做功能

在判断两个结构体变量是否类型等价时，需要去查符号表来获得这两个变量的 Type 信息，然后将这两个 Type 信息进行比较来判断是否相等，我们在 HelpFunc.c 中定义了一系列的辅助函数，其中的 IsTypeEqual（）函

数用来判断两个 Type 的类型是否相等。

四、 编译说明

编译时采用如下语句：

```
bison -d syntax.y
flex lexical.l
gcc main.c syntax.tab.c tree.c SemanticAnalysis.c SymbolTable.c HelpFunc.c -ly
-lfl -o parser
```