编译原理实验二：语义分析

——杨嘉兴+21311054+2807593076@qq.com

一、功能实现

1. 实现了所有的**必做**内容，即对 **17 种错误类型**都可以正确识别并且返回报错信息。

2. 实现了所有的**选作**内容，主要包括以下**三个要求：**

- **实现了**修改假设为“函数除了在定义之外还可以进行声明”，同时函数的定义仍然是不可以重复出现的，但是函数的声明在相互一致的情况下可以重复出现。同时我们还检查出了另外两种新的错误和增加新的产生式
- **实现了**修改假设为“变量的定义受可嵌套作用域的影响，外层语句块中定义的变量可在内层语句块中重复定义（但此时在内层语句块中就无法访问到外层语句块的同名变量），内层语句块中定义的变量到了外层语句块中不会消亡，不同函数体内定义的局部变量可以相互重名”。同时在该假设下，完成了错误类型 1 至 17 的检查。
- **实现了**修改假设为“将结构体间的类型等价机制由名等价改为结构等价”，同时在该新的假设下，完成了错误类型 1 至 17 的检查。

二、符号表的实现

1. 符号表的代码实现的头文件放在 **semantic.h** 中，具体的实现放在了源文件 **semantic.c** 中。

2. 符号表在语义分析中的作用是可以记录程序中定义的**符号（例如变量、函数、结构体等）**以及其**相关信息（例如类型、作用域、参数等）**。通过符号表，可以进行符号的**查找**、**插入**、**删除**等操作，并且可以判断两个类型的等价性。符号表在语义分析过程中用于检查语法的正确性、类型的一致性以及解析符号的作用域等。**符号表**是编译器实现中重要的数据结构之一，为后续的代码生成和优化提供了基础信息。

3. 我们实验的符号表实现主要采用了散列表和链表结构，符号表的**具体构建过程**如下：

- **初始化符号表：**在函数 initSymbolTable()中，首先将符号表的每个槽位初始化为空指针，并创建层次链表的头节点。
- **散列函数：**hash_pjw()函数使用了 PJW 算法，根据符号的名称计算散列值。该算法将每个字符的 ASCII 码左移两位后与当前散列值进行异或运算，并根据结果进行一些位操作，最终得到散列值。
- **向符号表中插入符号：**insertSymbol()函数通过散列值找到对应的槽位，在该槽位的链表头插入新的符号节点。同时，将新的符号节点插入到层次链表的头节点后面，以保持层次的顺序。
- **从符号表中查找符号（槽位）：**findSymbolAll()函数根据符号的名称计算散列值，并在对应的槽位链表中查找符号节点。
- **查找同一层次的符号：**findSymbolLayer()函数从层次链表中找到当前层次的节点，然后在该层次的链表中查找符号节点。
- **在符号表中查找函数符号：**findSymbolFunc()函数与 findSymbolAll()类似，但是只查找函数类型的符号节点。
- **从符号表中删除符号：**delSymbol()函数根据符号的名称找到对应的符号节点，并从符号表中删除。
- **插入一个层次：**pushLayer()函数创建一个新的层次节点，并将其插入到层次链表的

头节点之后。

- **弹出一个层次，同时删除该层次对应的所有符号：**popLayer()函数从层次链表中删除一个层次节点，并且删除该层次所对应的所有符号节点。
- **类型等价判断函数：**typeEqual()函数用于判断两个类型是否等价。它首先处理特殊情况，然后根据类型的种类进行不同的判断。对于基本类型，只有当两个类型都为基本类型且类型相同时才认为它们是等价的。对于数组类型，只有当元素类型是等价的才认为它们是等价的。对于结构体类型，只有当结构体成员的类型是一一对应等价的才认为它们是等价的。对于函数类型，需要判断返回类型、参数个数和参数类型是否等价。

三、在语法分析树上进行语义分析

1. 主体结构：实验 2 所有的语义分析都是基于实验 1 的语法树做的，宏观而言的工作是从语法树中获得信息，然后进行填表，查表分析；大部分调用的函数都是采用递归实现的，少部分需要进行额外链表链接的部分，比如 StructSpecifier 部分使用了循环实现，通过多个指针加上 while 的循环结构完成了 struct 中链表的组装工作。

2. 细化来说，在语法分析树上进行语义分析的大致实现过程可以**如下所示：**

- **初始化符号表：**在语义分析之前，需要初始化符号表，用于存储变量、函数等符号的信息。
- **程序入口函数：**该函数作为语义分析的入口，调用了 initSymbolTable()进行符号表的初始化，然后调用 Program()函数处理程序的根节点，最后调用 check()函数检查是否有函数声明但没有被实现的情况。
- **处理程序根节点：**调用 ExtDefList()处理外部定义列表节点。
- **检查函数被声明但没有被实现的情况：**遍历符号表中的每个条目，如果对应的条目的类型为函数，并且字段 hasDefined 为 0，表示存在已经声明但是没有被实现的函数，会输出错误信息。
- **处理外部定义列表节点：**该函数首先判断根节点是否有子节点，如果有，就依次处理第一个子节点（外部定义节点）和第二个子节点（外部定义列表节点）。
- **处理外部定义节点：**该函数首先调用 Specifier()函数获取类型信息，然后根据类型信息进行相应的处理。如果是结构体定义，则将结构体信息插入符号表；如果是全局变量定义，则调用 ExtDecList()处理全局变量名称列表；如果是函数定义，则调用 FunDec()处理函数声明，并将函数类型信息插入符号表。
- **类型描述符：**该函数根据传入的语法树节点确定相应的类型，并返回一个表示类型信息的结构体 Type。如果是基本类型（int 或 float），则创建一个基本类型的 Type 结构体；如果是结构体类型，则调用 StructSpecifier()处理结构体类型。
- **全局变量名称列表：**该函数用于处理全局变量名称列表，递归处理每个全局变量，并将其插入符号表。
- **函数声明处理：**处理函数声明或定义的函数头部信息，包括提取函数名、行号信息以及处理参数列表。根据语法树节点的子节点数目，判断是否有参数列表，并将参数的符号表条目插入符号表中。
- **复合语句处理：**处理函数体，包括变量定义和语句的执行。首先将函数参数存入符号表，然后调用函数处理函数体内的变量定义（DefList 函数）和语句（StmtList 函数）。
- **结构体声明或定义处理：**处理结构体声明或定义，包括设置结构体类型、处理结构

体名称、处理已定义的结构体类型以及处理结构体的成员变量定义（DefList 函数）。

- **变量的定义和声明处理：**递归地处理每个定义，并将结果连接成一个链表返回。调用 Def 函数处理单个定义。
- **单个定义处理：**处理变量的单个定义，包括调用 Specifier 函数获取类型信息，调用 DecList 函数处理声明列表。
- **声明列表处理：**递归地处理每个声明，并将结果连接成一个链表返回。调用 Dec 函数处理单个声明。
- **单个声明处理：**处理单个声明，包括调用 VarDec 函数处理变量声明，检查结构体域的初始化，以及检查变量声明的类型是否匹配。
- **变量声明处理：**根据节点的子节点数量，判断是变量声明还是数组声明。对于变量声明，插入符号表，并返回变量的符号表条目。对于数组声明，创建新的类型并递归调用 VarDec 函数。
- **函数参数列表的处理：**通过调用 ParamDec 函数处理函数的单个参数，获取参数的类型信息，并将参数加入到参数列表中。
- **语句列表的处理：**通过递归调用 Stmt 函数处理每一个语句，并按顺序执行。
- **单个语句的处理：**根据语句的类型进行不同的处理：
  - 如果是 **return 语句，**调用 Exp 函数处理返回表达式，并检查返回类型是否匹配。
  - 如果是**复合语句，**创建一个新的作用域，并调用 CompSt 函数处理复合语句的内容。
  - 如果是**表达式语句，**调用 Exp 函数进行处理。
  - 如果是**循环语句，**调用 Exp 函数处理循环条件，并递归处理循环体语句。
  - 如果是**条件语句，**调用 Exp 函数处理条件表达式，并递归处理 True 分支和可选的 False 分支的语句。
- 在抽象语法树上对表达式进行类型检查和语义分析，主要包括以下内容：
  - **对结构体使用"."操作符：**检查结构体类型是否正确，并检查访问的域是否存在。
  - **数组取地址操作：**检查操作数是否为数组类型，并检查索引是否为整数类型。
  - **赋值操作：**检查赋值操作的左值和右值的类型是否匹配。
  - **普通二元运算操作：**检查二元运算操作数的类型是否匹配，并根据运算符类型返回结果类型。
  - **表达式被括号包围：**递归处理括号内的表达式。
  - **一元减号操作：**检查操作数是否为基本类型。
  - **逻辑非操作：**检查操作数是否为整数类型。
  - **标识符（变量名或函数名）：**对于变量名，检查是否已定义并返回变量类型；对于函数名，检查是否已定义、参数是否匹配，并返回函数返回类型。
  - **INT 变量声明语句：**返回 INT 类型。
  - **FLOAT 变量声明语句：**返回 FLOAT 类型。

四、代码编译与运行

1. 代码编译：进入 Code 目录下执行 **make** 命令，会在该目录下产生相应可执行文件 parser。
2. 代码测试：编译完成得到 parser 文件后，通过**./parser ../Test/test1.cmm** 的形式进行相对应的测试文件进行测试。