编译原理 Lab2 实验报告

171860526，褚有刚ajeo0526@gmail.com　　171180554, 夏宇

2019 年 11 月 8 日

1　实验目的

在词法分析和语法分析的基础上编写一个程序，对 C–源代码进行语义分析和类型检查，并打印分析结果。

2　实验流程

2.1　符号表数据结构的实现

我们原来计划的是实现哈希表和红黑树组合的数据结构来使得符号的检索效率达到最优，但是在实现了红黑树之后，效率已经十分满意[1]，和哈希表组合创建更复杂的数据结构对于实际用来测试的代码量没有必要，而且可能拖慢检索效率，因此选择了原始的红黑树作为符号表底层的数据结构，为了使得红黑树的数据结构具有通用性，在每个节点中包含一个泛化的类型 **Entry**，并且使用从外界传入的比较器进行节点之间的比较，为了 debug 的方便，还提供了方便遍历的迭代器接口；

2.2　类型系统的实现

我们原来计划将类型系统独立于符号表存在，在类型系统中记录类型信息（数组类型，结构体类型，基本数据类型），然后在代码中出现的每个符号中持有类型的指针，但是由于用来测试的代码中出现的符号和类型的总数并不是很多，就将类型和符号合并在一个符号表中，使用枚举类型来区分符号和类型；

在代码中可能出现的符号有变量名，函数名和结构体类型名，对于变量，记录变量的类型和变量名；对于函数，记录函数的返回值类型和参数列表以及函数名，对于结构体，记录结构体的成员列表和结构体的名字（对于匿名结构体的处理见3.1），对于类型，由于基础数据类型的数量有限，所有在所有符号之间共享，对于数组类型，由于数组类型变化多，所有每个符号单独持有；对于结构体类型，由于其被设计为一种符号，所以在符号表中存储了完整的类型信息，对于其他符号只需要持有该类型的指针即可；

[1] 百万级别的数据只需要最多 40 次检索

2.3 递归进行的语义分析

对于语法分析树中的很多节点，是为了描述语言的结构而出现的，有一定的冗余性，如果对于每个文法符号都设计一个分析的过程，那么代码量会很多，在语义分析的过程中，事实上我们只需要关注一些**关键节点**[2]即可，比如 **DefList**, **ExtDefList** 需要向符号表中插入变量符号；**Specifier** 需要查询符号表或者向符号表中插入代表结构体类型的符号；**Statement**，**Exp** 中需要通过查询符号表来检查语句的正确性，在递归的过程中，遇到关键节点进行分析完成相应的语义动作动作即可；

3 遇到的问题和解决方案

3.1 棘手的内存管理

在设计符号表的结构的初期我们就考虑到了内存清理的问题，我们的设计思路是，结构体作为一个独立的符号，其他使用到结构体的地方持有该结构体引用，因此对于匿名结构体如果不插入符号表就不会被释放，因此，我们为每个匿名结构体生成一个**唯一不重复的名字**（使用数字的字符串表示，因为在 C–中数字不能作为标识符）以和有名结构体采用统一的处理方式；

3.2 连续的报错处理

在表达式分析的过程中，一个语义错误可能引发很多不必要的语义错误，我们的处理方式是在表达式分析过程一旦检测到语义错误就返回 NULL 值，而当 NULL 值被返回时，说明错误已经被报告，因此此错误可能引起的新错误将被忽视，继续向调用者返回 NULL 值，直到一个不是该错误引起的新的错误被检测到或者该条语句已经结束；

3.3 语法分析树的优化

在实验一的语法分析树的节点中使用字符串记录了每个节点的文法符号类型，在本次实验中，需要大量的根据文法符号的类型执行相应的代码的控制流语句，而字符串的比较是低效的，因此将语法分析树节点中原来表示文法符号类型的字符串转换为枚举类型，就能够在 **switch-case** 语句中根据文法符号中记录的表示类型的枚举值跳转到相应的处理代码处执行，**switch-case** 语句会使用编译器生成的**跳转表**，从而提高语义分析的效率；

此外在文法符号中增加记录子节点的数量的成员，在分析过程中根据产生式左部的文法符号的子节点数量就能够分析出产生式右部的组成而针对不同的产生式采取不同的语义动作。

3.4 编程中遇到的问题

1. 由于类库版本的问题，在遇到 **double free** 错误，程序不会被及时终止，调试过程中遇到了一些问题；

[2]语义动作中需要和符号表交互的节点

2. 对于 strcmp 的返回值，不同的类库有不同的版本，按照一个实现版本定义的符号比较函数作为比较器导致红黑树的相关操作出现失误；