Compiler Lab Report

刘嘉欣

2023 年 11 月 10 日

1 功能与实现

1.1 语义分析

采用” 面向错误类型编程”. 我思考了很久从何处开始, 发现从底层出发, 慢慢理解顶层部分是最好的: 即从底层出发从语法树的下部往上走, 最终将所有语法单元串起来, 对每个错误类型所在的位置进行语法分析检查. 这部分我的实现没有什么特别值得说的地方, 都是在” 螺旋上升式” 地理解实验 PPT 中为什么这样做 (如函数的每个参数的目的是什么), 并不是很难但需要注意很多细节, 写代码的时候需要头脑清晰额外小心. 其中 Exp 的实现最为复杂, 错误类型最多, 但位于最底层的位置, 依赖的其他部分最少, 所以我最先实现了这一部分, 过程中也慢慢了解到实验的要求. 一开始没理解 VarDec 为什么返回 FieldList, 后来发现是为了填充 STRUCTURE 的内容, 同理 VarList 和 Args 也返回了相应的列表类型来构造函数或检查函数的变量. 最终, 我的程序能识别十七种词法分析错误, 且引入了作用域.

1.2 符号表的实现 (包含选做内容)

首先实现了一个存放类型值的 open hashing 的散列表, 哈希表参考了课程 PPT 的实现, 但是其大小增大为 0x3fff. 然后着手符号表的设计, 参考了讲义的方式, 实现了一个基于十字链表和 open hashing 的散列表的 Imperative 符号表, 每次进入语句块则为这层语句块新建一个链表用来串联该层中新定义的变量; 离开时则顺着该层语句块的链表将本层定义的所有变量删除. 每次插入元素时, 总是将新插入的元素放到这个槽下挂的链表并加入该层对应链表的最后一个位置 **(这里其实可以优化一下, 插入到表头的, 离开语句块删除的时候也更不容易导致 Segmentation Fault.)**

2 编译方式

采用 Makefile 进行编译，执行的命令如下：(其中，syntaxTree.c 和 syntaxTree.h 保留了实验 1 语法树的定义以及相关操作;semantic.c 和 semantic.h 为语义分析的内容, 其中符号定义和符号表的定义在 symbol.c 和 symbol.h 中)

1. flex -o lex.yy.c lexical.l 使用 Flex 编译 lexical.l 文件，生成 lex.yy.c

2. bison -o syntax.tab.c -d -v syntax.y 使用 Flex 编译 syntax.y，生成 syntax.tab.c 和 syntax.tab.h 两个文件，其中 lexical.l 引用了 syntax.tab.h

3. gcc -c syntax.tab.c -o syntax.tab.o 编译 syntax.tab.c，生成 syntax.tab.o

4. gcc -std=c99 -c -o syntaxTree.o syntaxTree.c 编译 syntaxTree.c，生成 syntaxTree.o

5. gcc -std=c99 -c -o main.o main.c 编译 main.c，生成 main.o

6. **本次实验:** gcc -std=c99 -c -o symbol.o symbol.c 和 gcc -std=c99 -c -o semantic.o semantic.c 将所需的 symbol.c 和 semantic.c 文件编译生成 symbol.o 和 semantic.o

7. gcc -o ./parser syntax.tab.o syntaxTree.o main.o symbol.o semantic.o -lfl -ly 把 main.o 和 syntax.tab.o 和 syntaxTree.o 放到一起进行编译成最终的可执行文件 parser

3 个人心得

由于期中很多事情, 还因为某些原因回家了一趟, 留给本次实验的时间只有短短三天, 我还是个人组队, 实在是有点让我忙到焦头烂额了, 最后全部完成 (包括实验报告) 已经到晚上了, 离截止时间只有短短的几个小时. 我第一天看了实验讲义和助教的 PPT, 很大程度上参考了课程 PPT 的思路, 列出了大致的框架; 第二天开始在语法树上动笔, 完成了大部分词法分析的内容, 在 Struct Specifier 的实现上费了很多精力; 而第三天, 也就是最后一天, 短短的几小时就实现了一个符号表 (庆幸在后续 Debug 过程中这部分几乎没有 Bug 出现), 实现一些和作用域相关的东西, 然后就是漫长 Debug 过程了.

一开始并没有将选做的内容引入, 考虑最简单的情况; 再引入作用域范围进行语法分析. 测试过程中发现了很多 BUG, 我通过调试和打印输出解决了一些, 但是得分还是很低, 后来通过将 Struct 域中的变量设为外部作用域而通过了大部分的测试样例. 最后因为要实现”Imcomplete function type”(处理’int func(int x, int y);’ 的语法错误) 在 Lab1 中加了几行代码, 这才通过了所有测试样例.

实验过程中发现自己依赖的 PPT 上也有不少错误, 因为没有认真检查绕了不少弯路 (特别是 Type_check 函数, 因为没有加括号导致的 Bug, 还有关于 ARRAY 类型的判断导致 Segmentation Fault). 实验代码也并不完美, 缺少了一些内存的释放, 等有空补回来.