编译原理实验报告：语义分析

叶炜煜 131220030

叶思静 131220025

(南京大学 计算机科学与技术系, 南京 210093)

1 实现功能

1.1 功能简介

本章实验为实验二，任务是在词法分析和语法分析程序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。与实验一不同的是，实验二不再借助已有的工具，所有的任务都必须手写代码来完成。

2 功能实现简介

本次实验主要实现的是语义识别，根据第一次编译原理实验中得到的父子兄弟语法树，对语法树的每个节点进行语义分析，依照讲义之中的要求，对于十七个语法错误进行识别，若存在语义错误则打印出错误的行号，错误的类型，以及其详细信息。

语义分析识别过程如下：在 semantic.h 文件中存储建立的数据结构，主要是两个结构类型 struct singleSymbol 和 struct structure，分别存储变量和结构的信息，其中变量结构不仅存储变量，也一同存储了函数的信息。在.h 文件中包含了 semantic.c 中的函数声明。而 semantic.c 中就是对于语义的分析，首先在 semantic()函数中进行初始化，然后向函数中传入语法树的根节点进行语义分析。该点 c 文件的主题内容就是语义处理函数。

下面简要介绍下 semantic.c 中每个语义处理函数的简要流程，首先通过传入函数的参数，获得当前要处理的节点信息，以及该节点的子节点和兄弟节点的信息，然后对子节点依据其子节点以及子节点的兄弟节点的类型进行匹配，执行相应的语义处理函数，而每个语义处理函数依据实验一规约的非终结符命名，其中的处理流程也完全依据附录中的规约规则。在规约过程中，依照十七个错误类型，在相应的位置进行检测与判断，若存在语义错误则输出错误信息，并返回上层，这样可以保障在识别到错误之后对于后续的语义继续进行分析。

当执行完所有的语义处理函数，则完成了全部的语义分析，屏幕中将显示测试文件中包含的错误信息，因为实验二为语义分析，所以是在测试文件中不包含词法和语法错误的前提下，对于代码的语义分析。

3 编译环境及编译过程

3.1 编译环境 Ubuntu12.04.2 flex bison

3.2 编译过程：进入程序所在文件夹 键入：make 然后键入 ./praser + 目标文件

4 创新

4.1 鉴于变量和函数所需要存储的内容与相关标记很相似，所以将变量和函数存储在相同的结构类型中，使用了一个union类型存储两者的具体信息，除此之外的结构成员都相同，良好的分装性使得我们在代码的书写中保持了较好的风格，同时也帮助我们较简单的理解了语义分析的逻辑框架。

4.2 在代码书写过程中，为了方便之后的调整与扩展，我们统一将所有的语义处理函数都设置为空，同时函数的参数列表中传入相关结构的指针，这样可以将当前函数中处理得到的信息传入到上一个函数中，并且也可以将信息继续往下传，为代码的书写带来了相当大的便利。

4.3 采用case原则处理语义错误分析，对于同种的错误一起分析，譬如函数重定义和变量重定义的过程相似，所以一起书写，然后保证该错误都能识别的情况下进行下个错误的识别。

4.4 对于选做部分2.2，我们采取了对每个变量建立深度变量，初始的时候深度都为0，在遇到左括号的时候将深度加一，然后遇到右括号的时候将深度减一，同时将当前深度下的变量从变量表中删除。在识别重定义错误的时候，若当前变量与变量表中的某一变量同名，此时还需要比较深度，若新识别出的变量深度较大，则为重定义错误。

5 遇到问题

5.1 本次代码的完成顺序为框架的搭建，即先写好所有的语义识别函数的框架，在后续补充，所以在后续补充的时候需要对之前所写的代码进行大量的重构，这一过程较繁琐也极易容易出错。并且因为完成的顺序是先识别变量相关的错误，然后识别函数相关错误，然后是数组错误，最后是结果相关的错误，因为是分阶段写，所以很容易导致两者结合的时候出现一些错误。

5.2 本次实验大量使用了指针，而指针悬停很容易出现段错误，所以在实验过程中大量出现了段错误，对于段错误的纠错也相当的麻烦，后来借助输出相关信息，对于错误的位置进行筛选，从而最终解决了遇到的所有的段错误。