编译原理 — Lab2

姓名：郑奘巍　　学号：171860658　　日期：2020 年 4 月 12 日

1 基本功能

本次实验完成了基本要求和要求 2.1, 要求 2.2, 要求 2.3。程序在生成的词法树基础上，通过遍历词法树（递归方式）进行属性文法的实现。具体实现如下：

symbol.c 中实现了基于哈希的符号表，使用链表解决碰撞冲突。使用了作用域栈（**compst**）支持作用域嵌套，从全局域为 0 层开始，每层作用域的层号 +1，第 i 层作用域内的变量以链表形式储存在 **compst[i]** 中。每次退出作用域时，将该层作用域变量从哈希表中删去（实现中打上无效标记）。寻找符号时，选择同名符号中层号标记最大的且有效的变量。

semantic.c 中实现了语义的检测。根据产生式规则，将每个符号作为函数实现属性文法。实现时先使正确程序能够跑通，并输出正确的符号表，再逐个添加错误类型进行检测。

最后，该程序通过持续集成的大量测试用例保证一定的正确性。

2 编译方法

本次实现共在三组环境中测试，使用的 gcc, flex, bison 分别为以下系统中支持的最新版：

1. 开发环境：macOS 10.15.4
2. 集成环境：Ubuntu 18.04
3. 测试环境：Ubuntu 12.04

编译方法：

1. 通过框架代码的 **Code/Makefile** 生成二进制文件 parser
2. 为了便于开发和测试，使用 **Lab/makefile** 生成二进制 parser，助教测试时可以无视该文件的存在。

3 实验亮点

除了实现基本功能和附加的三个要求之外，在本次实验中还有如下亮点：

- 易读易维护的代码

代码实现中，将数据结构和语义分析分别定义在 **symbol.c** 和 **semantic.c** 中，且语义分析严格按照语法规则逐步遍历，并用注释标记了对应的语法规则和错误类型检测。通过有效复用，两个文件代码行数之和仅 800 多，并包含进 100 行的注释。

通过 git 记录可以发现，在代码编写中，通过先搭建好属性文法框架，逐步添加错误类型，可以较迅速的完成实验二。

- 持续集成

  受董杨静同学 **compilers-tests** 项目的启发，我使用了自己的持续集成框架 **CI4C–_Compiler**。相比于原项目，该框架可以同时在检测 Lab2 样例的基础上同时检测原有的 Lab1 样例。该工具支持更多的参数，并以实验（Lab）-测试集（test）-测试样例的结构整理测试数据。在使用 Github 集成的同时，可以在本地也进行快速的测试。

  通过该方法可以有效的检测出实验中的问题，保证实现的正确性。

- 结构体等价的实现

  在结构体等价的实现中，首先判断了两者是否重名，加快了判断的速度。