编译原理第三次实验报告

姓名：李博文　　学号：2022111189

一、实验内容

任务是在词法分析、语法分析和语义分析程序的基础上，将 C 源代码翻译为中间代码

二、开发环境与编译方式

-开发环境：macOS+Clion+GNUFlex&Bison

-编译命令：在 Code/目录下执行 make 即可完成编译，详细操作方法见第六节

三、程序实现功能与原理

本实验在实验二的基础上实现中间代码生成，将 C--语言源代码翻译为三地址代码，并支持基本语句和表达式的中间代码输出。采用树形结构的语法分析结果，通过设计翻译函数，逐节点转换为线性中间代码。

四、程序模块结构介绍

lexical.l

负责词法分析，通过 Flex 工具生成词法分析程序，识别源代码中的 token 并传递给语法分析模块。

syntax.y

负责语法分析，通过 Bison 生成解析器，同时构建语法树，为后续中间代码生成提供树结构数据。

semantic.c/.h

负责语义分析，检查源程序语义错误并构建符号表，供中间代码生成时进行类型检查和变量作用域管理。

intercode.c/.h

核心模块，负责中间代码的生成与管理，提供多种翻译函数，如 translate_Exp、translate_Stmt 等，将语法树节点逐步转换成中间代码。

tree.c/.h

定义语法树节点结构，提供创建节点和树的遍历功能，供语法分析和中间代码生成模块调用。

main.c

程序入口，负责命令行参数解析，协调调用词法分析、语法分析、语义分析和中间代码生成模块。

五、具体实现细节

中间代码结构设计

采用线性中间代码（链表形式）保存，每条中间代码包括操作类型、操作数等信息，便于后续优化和输出。

翻译函数实现

为每种语法结构编写专门的翻译函数，递归遍历语法树节点，根据不同节点类型生成相应的中间代码片段，最终合并为完整的中间代码序列。

变量与标号管理

使用统一的接口动态生成临时变量和标号，保证代码正确性和可读性。

六、程序编译方法

1. 确保已安装 GCC、Flex 和 Bison 工具，且环境变量已正确配置。
2. 打开终端进入源码目录（含 Makefile 位置），运行以下命令编译程序：
3. make
4. 编译完成后，生成可执行文件 parser，执行以下命令运行程序：
5. ./parser test1.cmm out1.ir
6. 上述命令中，test1.cmm 为输入的 C--语言源程序文件，out1.ir 为输出的中间代码文件。

五、实验亮点与总结

· 中间代码结构合理，支持多样语法结构

采用线性结构的三地址码表示，涵盖赋值、算术运算、条件跳转、函数调用、输入输出等常见结构，基本满足 C–语言的表达能力，同时兼容 IR Simulator 运行环境。

· 标号与临时变量自动生成机制

使用统一计数器自动管理临时变量名和标号命名，确保不同作用域下中间代码的唯一性和正确性，避免命名冲突。

六、实验体会

本次实验在前两次词法与语法分析基础上，完整实现了一个将 C–语言翻译为三地址中间代码的编译器前端核心模块。通过递归遍历语法树并设计一系列翻译模板，成功将抽象语法结构映射为线性 IR 表示，并能够在虚拟机环境下成功执行。

实验不仅加深了对编译原理中间代码层的理解，也锻炼了将理论应用于工程实践的能力。