编译原理实验阶段 3 实验报告

一、实验完成情况

本次阶段的必做、选做部分都已完成，能够对输入的代码文件进行中间代码生成并将代码生成结果输出，输出的代码经过 irsim 检查可编译成功。中间代码的生成规律参照了阶段 3 指导的中间代码形式的表格来生成。

二、实验运行操作

由于 flex 文件与 bison 文件没有变化所以不用再次运行。

运行

gcc main.c c_analyse.c common.c inter_code.c operand.c symbol.c syntax.tab.c syntax_tree.c translate.c -lfl -ly -o parser

来生成可执行文件 parser，然后执行

./parser 输入文件名 output.txt

将输入代码文件生成中间代码并将生成结果输入到 output.txt 中，然后将 output.txt 的文件名改为 output.ir 后就可以使用 irsim 来进行中间代码的检测。

三、实验思考

本次实验由于需要再次实现对语法树的遍历操作所以需要实现类似于阶段 2 中的分析代码的翻译代码，所以我将阶段 3 实验指导中给

到的基本表达式的翻译代码、语句的翻译代码、条件表达式的翻译代码、函数调用的翻译代码以及函数参数的翻译代码都与其他翻译步骤整合在 translate.c 代码中。在将基本的翻译操作都实现完后我也把内部代码和处理数的结构体构造完成，但运行时发现中间代码漏掉了很多，我反复检查了才发现忘了把 read 和 write 作为符号添加到符号表中，于是又让我调了很久代码。本次代码的处理数的实现上，我用了一个静态的整型数来表示 id，并每当有处理数的创造时让它加 1，这样能保证处理数的 id 不重复且能逐个上升。在翻译的连锁函数的实现中，由于与语义分析时的代码及其相似，导致我一开始编写代码时将条件表达式翻译函数完全跳了过去，后来也调了很久代码才发现我把条件表达式翻译函数漏掉了。