编译原理实验报告三

代码功能

1. 根据语法树，将较为简单的代码转化成中间代码
2. 对有结构体的代码转化成中间代码
3. 对有函数的代码转化成中间代码

代码完成部分

1. 优化加法函数

   当两个操作数都是常量时，我们可以直接得到两个操作数之和，就不需要专门生成一条表达式来计算存储这两个操作数的和

   当其中一个操作数为0，而另一个操作数不为取地址和解引用操作数，结果就可以直接等于非零操作数

2. 优化乘法函数

   乘法和加法的类型类似，同样需要讨论两个操作数均为常数，其中一个操作数为0的情况。

   此外，乘法还需要讨论其中一个操作数为1的情况，此时结果直接等于另一个不等于1的操作数

3. 基本表达式翻译函数中，以单个变量为左值的情况

   首先先建立一个操作数来存储左值，再建立一个操作数来存储右值，因为左值为单变量而右值有可能是表达式，所以我们需要提前通过基本表达式翻译函数翻译右式，最后建立中间等式代码即可

4. 基本表达式翻译函数中，基本表达式为条件表达式的情况

   SDT为：

Exp_1 RELOP Exp_2 NOT Exp_1 Exp_1 AND Exp_2 Exp_1 OR Exp_2	label1 = new_label() label2 = new_label() code0 = [place := #0] code1 = translate_Cond(Exp, label1, label2, sym_table) code2 = [LABEL label1] + [place := #1] return code0 + code1 + code2 + [LABEL label2]

5. 条件表达式的翻译函数

   SDT为：

translate_Cond(Exp, label_true, label_false, sym_table) = case Exp of	
Exp_1 RELOP Exp_2	t1 = new_temp() t2 = new_temp() code1 = translate_Exp(Exp_1, sym_table, t1) code2 = translate_Exp(Exp_2, sym_table, t2) op = get_relop(RELOP); code3 = [IF t1 op t2 GOTO label_true] return code1 + code2 + code3 + [GOTO label_false]
NOT Exp_1	return translate_Cond(Exp_1, label_false, label_true, sym_table)
Exp_1 AND Exp_2	label1 = new_label() code1 = translate_Cond(Exp_1, label1, label_false, sym_table) code2 = translate_Cond(Exp_2, label_true, label_false, sym_table) return code1 + [LABEL label1] + code2
Exp_1 OR Exp_2	label1 = new_label() code1 = translate_Cond(Exp_1, label_true, label1, sym_table) code2 = translate_Cond(Exp_2, label_true, label_false, sym_table) return code1 + [LABEL label1] + code2
(other cases)	t1 = new_temp() code1 = translate_Exp(Exp, sym_table, t1) code2 = [IF t1 != #0 GOTO label_true] return code1 + code2 + [GOTO label_false]

   所以在条件表达式翻译函数中，需要通过分类来对每种情况来生成中间代码即可

代码结构

代码中包括了10个文件，其中 **intercode.c** 和 **intercode.h** 用于生成中间代码

intercode.h 定义了用于生成中间代码的函数，以及操作数的结构体，指令的结构体

intercode.c 包括了对 **intercode.h** 中定义的函数的实现过程

代码运行

```
make clean // 清除编译文件

make  // 生成编译文件

make test // 对测试文件进行测试
```

其中，在 `Makefile` 中，我们可以在下面图片中圈出的代码处填入测试文件的相对路径，执行完 `make test` 后，文件夹会生成一个 `out.ir` 文件，该文件就可以通过虚拟机的小程序模拟中间

代码过程

```
45    .PHONY: clean test
46    test: parser_
47            ./parser ./test_code1.txt out.ir
```