编译原理实验报告

姓名：张旭钦

学号：201240058

中间代码为指令集，而指令的基本组成单元为操作数（即讲义中的Operand）。考虑到中间代码的可读性和易调试性，对于源代码出现的变量，它们在中间代码中以`v%d`的格式存在；而中间生成的临时变量，不管它们是存储值还是地址，都以`t%d`的形式存在。

# 问题合集

问题一：如何区分函数参数和局部作用域中声明的变量

实验三要求当函数参数为数组或者结构体时，传递的为地址。在`translate_exp`中查找`id`时，存放地址的`id`和存放值的`id`在生成中间代码的行为有所不同，因此对于函数参数，插入符号表时我将其标记为`OP_ADDR`；对于局部作用域我则将其标记为`OP_VARIABEL`。

问题二：如何区分临时变量中存储的是值还是变量

Operand绑定了一个标记变量，=0时表示存储值，=1时表示存储地址。至于为什么要将两种临时变量加以区分，可以看后面的赋值表达式的处理函数。

```
// 我们约定临时变量可能存储值也可能存储地址，当tmp_addr = 1时表示存储地址，否则存储值
#define IS_TMP_ADDR(t) (t->kind == OP_TEMP && t->u.tmp.tmp_addr == 1)
// 临时变量存储值
#define IS_TMP_VAL(t) (t->kind == OP_TEMP && t->u.tmp.tmp_addr == 0)
```

问题三：赋值表达式的翻译模式如何分析

```
/// @brief Exp -> Exp ASSIGN Exp (表达式左部有三种情况: ID, Exp[Exp],
/// Exp.ID)，所以本函数直接根据表达式左部展开式分为三类情况进行分析
/// @param exp ASTNode节点，place依旧是需要返回的Operand
void exp_assign(ASTNode *exp, Operand place) {
    Operand left = new_tmp_op();
    Operand right = new_tmp_op();

    if (match(exp->child_list[0], 2, "Exp", "ID")) {
        translate_exp(exp->child_list[0], left);
        translate_exp(exp->child_list[2], right);
        insert_assign_ir(ASS_NORMAL, left, right); // left := right
    } else if (match(exp->child_list[0], 4, "Exp", "Exp", "DOT", "ID")) {
        // 左边为结构变量的域时，我们希望解析左边表达式能够返回一个地址，这样我们才可以使用 *left =
right
        left->u.tmp.tmp_addr = 1;
        translate_exp(exp->child_list[0], left);
        translate_exp(exp->child_list[2], right);
        insert_assign_ir(ASS_SETVAL, left, right); // *left := right
    } else {
        // 左边为数组成员，我们希望解析左边的表达式能够返回一个地址，这样我们才可以使用 *left = right
        left->u.tmp.tmp_addr = 1;  // we are suppose to obtain the addr of the left exp
        translate_exp(exp->child_list[0], left);
        translate_exp(exp->child_list[2], right);
        insert_assign_ir(ASS_SETVAL, left, right);
    }

    // 因为不会出现数组和结构体的直接赋值，因此正确
    if (place) {
```

```
        // 这里非常精妙，需要仔细思考和斟酌
        // 主要是考虑到多个赋值连续的情况，注意赋值号解析是从右往左的
        int left_is_addr = IS_TMP_ADDR(left) || (left->kind == OP_ADDRESS);
        if (!left_is_addr)  // 如果是普通变量的赋值，left存放值，place := left
            insert_assign_ir(ASS_NORMAL, place, left);
        else  // 如果是数组或者结构体赋值，那么left存放地址，需要place := *left
            insert_assign_ir(ASS_GETVAL, place, left);
    }
}
```

## 维护和调试

为了易于维护和调试代码，我将实验三的内容独立地实现在`intercode.h`和`intercode.c`文件。

借助于`vsocde`调试环境，代码中出现地段错误和死循环或者中间代码生成错误的问题可以被方便地找出。

测试代码正确性时，我使用了针对性的单元测试，如果单元测试没有通过，通过中间指令和额外的`write`先大致定位错误，然后再最小化的复现错误，最终定位到源代码中的错误。

比如，在某一次调试中，我的错误如下：

结构体内部成员包含数组时，当结构体内的数组作为实参传递给函数时，我传递的是数组的值而非数组的地址。

通过观察中间代码和断点调试，我在分析结构体成员时多加了一个判断：只有当结构体中的域为基本类型时才需要返回值，否则临时变量存储成员的地址。由此就可以解决之前的问题的。