

LAB 3 实验报告

裴明亮 151242033

一. 实验环境及编，译方法

使用的是 Windows 系统（安装了虚拟机），在实验的根目录下创建了 Makefile 文件：进行测试时，执行 Make test 即可执行相应测试样例的输出。

二. 文件结构

在 lab2 的基础上添加了新增了 intercode.c 和 intercode.h, 主要实现与中间代码生成相关的数据包

相关宏定义;

```
#define VAR_OP 0x101    //表示变量类型的操作数
#define FUNC_OP 0x102   //表示函数标号类型的操作数
#define LABEL_OP 0x103  //表示标号类型的操作数
#define CONST_OP 0x104  //表示常量类型的操作数
#define TEMP_OP 0x105   //表示临时变量类型的操作数

#define LABEL_CODE 0x1   //表示标号代码
#define FUNCTION_CODE 0x2 //表示函数声明代码
#define ASSIGN_CODE 0x3  //表示赋值代码
#define ADD_CODE 0x4      //表示加法代码
#define MINUS_CODE 0x5    //表示减法代码
#define MUL_CODE 0x6      //表示乘法代码
#define DIV_CODE 0x7      //表示除法代码

#define ADDRESS_CODE 0x8  //表示地址代码

#define GOTO_CODE 0xb     //表示 GOTO 跳转语句代码
#define IF_GOTO_CODE 0xc  //表示条件跳转语句代码
#define RETURN_CODE 0xd   //表示返回语句代码
#define DEC_CODE 0xe      //表示申请空间代码
#define ARG_CODE 0xf      //表示参数代码
#define CALL_CODE 0x10    //表示函数调用代码
#define PARAM_CODE 0x11   //表示形参代码
#define READ_CODE 0x12    //表示 read 函数
#define WRITE_CODE 0x13   //表示 write 函数
```

三. 数据结构

新增两个数据结构

1. operand 表示操作数的结构

```
struct _operand
{
    int kind;//指示该操作数是什么类型
    union
    {
        //标号类型，常量类型，临时变量类型使用
        int no;
        //变量类型，函数类型使用
        char *value;
        //地址类型使用
        operand *addr;
    };
    //参数列表中的下一个参数
    operand *nextArg;
};
```

2. code 表示中间代码结构

```
struct _code
{
    int kind;//指示该代码的类型
    union
    {
        //useless 只是无用代码，为了数据对齐
        struct
        {
            operand *left;
            operand *right;
            operand *useless;
        }assign;//赋值操作代码，left 表示等号左值，right 表示等号右值

        struct
        {
            operand *useless1;
            operand *op;
            operand *useless2;
        }singleop;//单操作数的代码如：标号，函数声明，read，write

        struct
        {
            operand *result;
```

```

        operand *op1;
        operand *op2;
    }doubleop;//双操作数的代码，如：加法，减法，乘法，除法等。

    struct
    {
        operand *gotoLabel;
        operand *x;
        operand *y;
    }tripleop;//三变量的代码，如：if 语句

    struct
    {
        operand *op;
        int size;
    }dec;//空间申请代码
}detail;
char *relop;//若为 relop 操作，则该变量指代是何种 relop
code *pre;//链表的前一个结点
code *next;//链表的后一个结点
};

```

四. 翻译函数

```

void printcode(char *outfilename);//往文件中保存中间代码
void insertCode(code *p);//将代码插入当前代码链表之后
void translate_Basic_Exp(Node *root, operand *place);//翻译表达式
void translate_Args(Node *root, operand **arglist);//翻译参数列表
void translate_Stmt(Node *root);//翻译声明
void translate_Cond(Node *root, operand *label_true, operand *label_false);//翻译条件语
句
void translate_CompSt(Node *root);//翻译 CompSt 语句
void translate_DefList(Node *root);//翻译定义语句
void translate_Def(Node *root);//翻译定义语句
void translate_DeclList(Node *root);//翻译定义语句
void translate_StmtList(Node *root);//翻译声明语句
void translate_Dec(Node *root);//翻译定义语句

```

四. 总结

按照试验指导给出的表格进行函数分情况讨论并具体实现。主要完成中间代码生成和中间代码优化，基本完成必做部分。