lab3

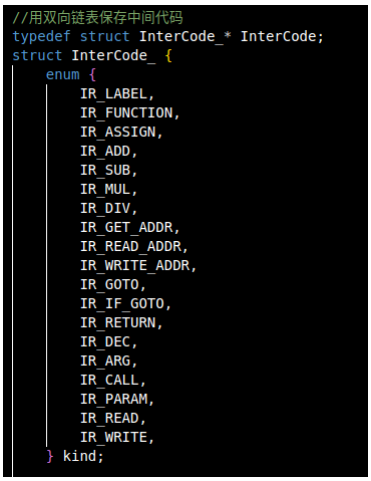
211275009 陈铭浩
211275009@smail.nju.edu.cn

实现功能

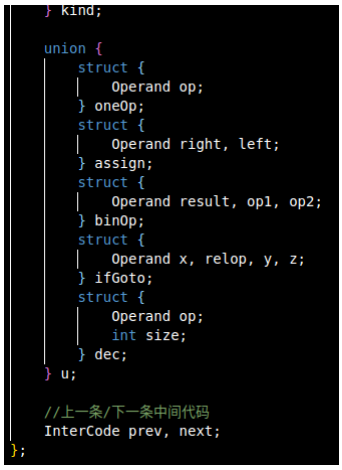
- 在lab1和lab2的基础上，将C--语言的源代码翻译为中间代码，符合讲义中给出的中间代码形式及操作规范。
- 完成选做2：一维数组类型的变量可以作为函数参数（但函数不会返回一维数组类型的值）;可以出现高维数组类型的变量（但高维数组类型的变量不会作为函数的参数或返回类值）。

实现思路

1. 首先需要确定中间代码的组织形式，这里我采用了双向链表的线性表示方法，具体结构定义如下



```
//用双向链表保存中间代码
typedef struct InterCode_* InterCode;
struct InterCode_ {
    enum {
        IR_LABEL,
        IR_FUNCTION,
        IR_ASSIGN,
        IR_ADD,
        IR_SUB,
        IR_MUL,
        IR_DIV,
        IR_GET_ADDR,
        IR_READ_ADDR,
        IR_WRITE_ADDR,
        IR_GOTO,
        IR_IF_GOTO,
        IR_RETURN,
        IR_DEC,
        IR_ARG,
        IR_CALL,
        IR_PARAM,
        IR_READ,
        IR_WRITE,
    } kind;
```



```
    } kind;

    union {
        struct {
            Operand op;
        } oneOp;
        struct {
            Operand right, left;
        } assign;
        struct {
            Operand result, op1, op2;
        } binOp;
        struct {
            Operand x, relop, y, z;
        } ifGoto;
        struct {
            Operand op;
            int size;
        } dec;
    } u;

    //上一条/下一条中间代码
    InterCode prev, next;
};
```

2. 此外，我还封装了操作数Operand以及函数参数列表ArgList
3. 生成中间代码的过程与lab2相似，都是遍历语法分析树，在对应的函数中生成对应中间代码，最后按照规定格式将中间代码打印即可
4. 本实验最折磨我的部分是选做部分。一方面，将高维数组引入后，在翻译数组时，需要额外增加一个遍历，找到数组最内层的ID，一边查询符号表并计算出相应的内存地址以便使用。另一方面，当一维数组类型的变量可以作为函数参数时，就需要传引用（数组的首地址）以实现参数传递，这里需要在处理实参时，进行特判，同时也需要特判形参是一维数组的情况，保证使用时将传入的一维数组直接当作地址使用。此外也需要特判是否迭代调用了一维数组参数，如果迭代了，则该参数已经是地址不需要再取地址。

```
while (argTemp) {
    //printf("debug\n");
    //printf("argtemp_op: %d\n", argTemp->op->kind);
    if (argTemp->op->kind == OP_VARIABLE) {
        SymbolTableEntry* item = find(argTemp->op->u.name, 2);
        //printf("debug\n");

        // 一维数组作为参数需要传址
        if (item && item->type->kind == ARRAY) {
            //区分是否是迭代调用了一维数组参数，如果迭代了，已经是地址不需要再取地址
            Operand varTemp = newTemp();
            if(item->isArg==1){
                genInterCode(IR_ASSIGN, varTemp, argTemp->op);
            }else{
                genInterCode(IR_GET_ADDR, varTemp, argTemp->op);
            }
            Operand varTempCopy = (Operand)malloc(sizeof(struct Operand_));
            varTempCopy->kind = OP_VARIABLE;
            varTempCopy->u.name=my_strdup(varTemp->u.name);
            // varTempCopy->isAddr = TRUE;
            genInterCode(IR_ARG, varTempCopy);
        }
        // 一般参数直接传值
        else {
            genInterCode(IR_ARG, argTemp->op);
        }
    } else {
        genInterCode(IR_ARG, argTemp->op);
    }
    argTemp = argTemp->next;
}
```

5. lab3个人认为比前两个实验都要复杂，一方面debug不便（主要通过printf），另一方面庞大的代码量让我最后改bug时深刻感受到了什么是屎山，这也警醒我要更注意代码的规范性和数据结构设计合理的重要性

编译方式

```
在Lab3/Code文件夹下
$ make parser
然后用Code/parser替换Lab3下的parser（当然，在提交的版本中我已经替换过了）
在Lab3文件夹下执行
$ ./parser <输入文件路径> //控制台输出
或
$ ./parser <输入文件路径> <输出文件路径> //输出到单独文件中
```