# Lab 2: Dataflow and optimizations

小组成员：张崴，刘梦馨

1. **实验完成目标：**
   1. Construct the Control Flow Graph (CFG).
   2. Perform Strongly Connected Region (SCR) analysis.
   3. Perform reaching definitions and use it to perform simple constant propagation.
   4. Perform live variable analysis and use it to perform dead statement elimination.
2. **程序使用方法**

执行complish.sh完成程序编译。run.sh后加参数执行相应程序，-backend后接c,cfg,scr，3addr,rep分别表示生成c代码，cfg报告，scr报告，3addr代码和优化报告；-opt 后接scp和dse表示运行简单常量传播和死表达式消除。用重定向符’<’ 将指定程序3addr代码交给相应程序执行。

1. **程序设计思想**
2. 生成cfg
3. 找出所有的basic block 的起始点，分别是enter指令，blbc、blbs和br的跳转指令和紧接在blbc、blbs和br的下一条指令以及call的下一条指令。
4. 由于每个basic block只在最后一条语句有何别的block有边的关系，所以只需要考虑每个basic block的最后一条语句如果最后一条语句是br则边连接到目标指令，若最后一条语句为blbc或blbs则边连接到目标指令和下一条指令，若最后一条语句为call 边连接到下一条语句。
5. 以每个ret指令为一次分析的终点，输出分析结果。
6. 生成scr

寻找scr才用寻找nature loop的算法，首先找到一条从后向前跳转的指令，将这两个点插入集合，将其他和编号较大指令相连且大于循环开始的指令入栈，在寻找可到达栈中指令且不通过循环起始点的指令，将栈中指令出栈，新指令入栈，直至栈空。

由于原语言集合不包含goto语句，所以生成的程序中不会出现循环相互交叉的现象，最终循环之间的关系只会是相互独立或者相互包含。在这种没有交叉的情况下scr和nature loop是等价的所以找到了nature loop即为找到了scr。

所以如果两个scr之间包含相同的basic block那么较大的scr一定包含较小的scr只需要保留较大的scr。

1. 计算reaching definition
2. 首先生成类似于lab1中的中间代码，寻找到所有的赋值表达式，存入结构体Def数组，结构体记录定义变量名，定义变量的赋值，是否为常数以及在第几行代码定义。
3. 寻找赋值语句的kill关系，对同一变量的赋值将会kill掉其他位置的赋值，将生成集合存入结构体中。
4. 针对每条语句生成in集合和out集合，遇到跳转语句将out集合输出到对应语句的in集合做并。
5. 若每个out集合大小不再变化计算停止，否则再次循环计算reaching definition。
6. 常量传播

获取每条3addr的地址部分对应的变量，查看在该指令的in集合对应的变量处的所有定义值是否相等且为常数，如果相等且为常数，将该指令位置的变量替换为常数。

1. 计算live variable

才用和计算reaching definition类似的算法从最后一条语句开始向上执行，在每个basic block第一条语句时寻找是否有其他语句可以到达这条指令，将这条指令的in集合输出到可到达这条指令的out集合做并，循环算法直到各个in集合不再变化。

1. 进行死表达式消除

寻找在一个cfg内live一直不存在的变量，将该变量赋值的3addr语句改为nop