Lisa Chen
862186585
Project 3: Liveness Analysis

Algorithm/Code Summary

To ensure the variables' names work, I had done the following changes to the terminal commands:

- `opt -instnamer -load ../Pass/build/libLLVMValueNumberingPass.so -ValueNumbering 1.ll`
  - This code is simply called similar to the project for HelloPass, but in HelloDataFlow/Test folder.

File Structure and Compilation

1. There is a Block object created to represent each basic block and all the information we will need to generate: UEVar, VarKill, successors, predecessors. This makes it easier to do the computation later as we can access the information at any time during the worklist algorithm.
2. There are also a vector representing the worklist and a vector representing all the blocks (to use to search block names and add blocks to add to the worklist.
3. First Pass – Generating UEVar, VarKill, Successors, Predecessors
   a. The whole code is only passed through once for the UEVar, VarKill, list of successors, and list of predecessors for each block. In addition, we keep a list of all the blocks and generate the worklist with all the blocks on this pass.
      i. The worklist algorithm is separate from this pass. See Step 4.
   b. List of successors is simply from getting the last line in a block (using BranchInst getTerminator()) and find its operands, if it exists. This is where the branch operation is when they exist. We then store the operands.
   c. As I did not see a way in LLVM to have predecessors, I kept track of the predecessors using a predecessor hash map with a vector of predecessor blocks as the value to a block's name as a key. When we find a successors to a block (call this block A), we take each successor, get its predecessor list (if exists, or create one) and add block A to it.
   d. Because of -instnamer, we have %tmp variables created. To account for this, a var hashmap is created to link %tmp to the correct variable it represents. I then search the hashmap using getEquivVar() to find what the %tmp variable corresponds to.
   e. UEVar is added per block for binary operations and for certain loads.
      i. For example, for x <- y op z, y and z are added to UEVar (if it isn't already in VarKill).
      ii. In addition, if we have %tmp1 = load i32, i32* %c, align 4, c is also added to UEVar (if not in VarKill) to ensure that we capture all cases where a variable is about to be used.
         1. For example, e=a can cause a case that if we did not do this, we would not have a in UEVar).
   f. VarKill is added for binary operations. This is when we have x <- y op z, we simply add x to the VarKill vector.
4. Worklist

Lisa Chen
862186585
Project 3: Liveness Analysis

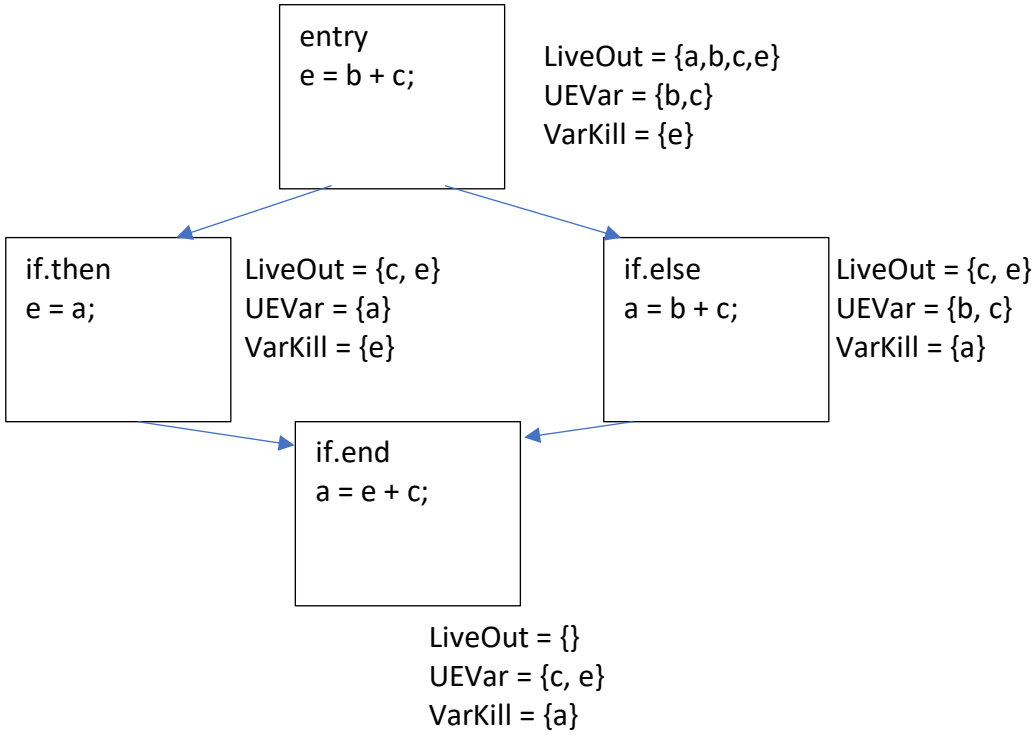
a. We take the initial worklist with all the blocks that was created in Step 3. Then, we will loop through the worklist until it's empty, popping block by block.
b. For each block popped, we compute the LiveOut using the equation in class:
$$LiveOut(n) = \cup_{x \epsilon succ(n)} (LiveOut(x) - VarKill(x) \cup UEVar(x)$$
c. If the worklist is changed (determined by if LiveOut vector size is changed since LiveOut can only get bigger), we add the predecessors to the worklist (if it isn't already in the worklist).
    i. We add uniquely to prevent unnecessary computations.

See Next page for test runs.

Lisa Chen
862186585
Project 3: Liveness Analysis

Test File 1.ll

```
Block name: entry
LiveOut
e c b a
UEVar
b c
VarKill
e

Block name: if.then
LiveOut
e c
UEVar
a
VarKill
e

Block name: if.else
LiveOut
e c
UEVar
b c
VarKill
a

Block name: if.end
LiveOut

UEVar
e c
VarKill
a
```

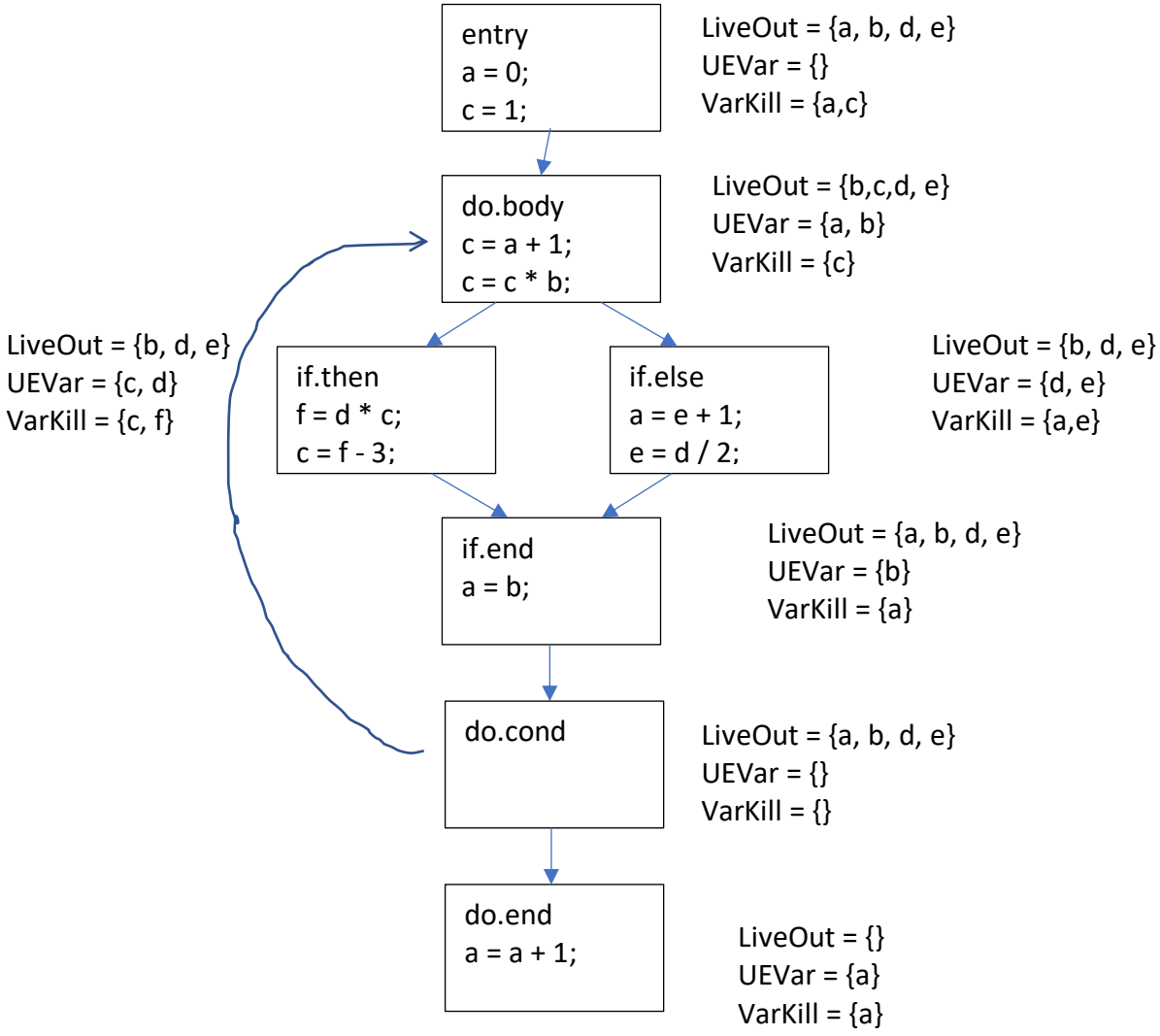


Lisa Chen
862186585
Project 3: Liveness Analysis

Test File 2.ll

```
Block name: entry
LiveOut
e d a b
UEVar

VarKill
a c

Block name: do.body
LiveOut
b e d c
UEVar
a b
VarKill
c

Block name: if.then
LiveOut
e d b
UEVar
d c
VarKill
f c

Block name: if.else
LiveOut
e d b
UEVar
e d
VarKill
a e
```

```
Block name: if.end
LiveOut
a e d b
UEVar
b
VarKill
a

Block name: do.cond
LiveOut
a e d b
UEVar

VarKill


Block name: do.end
LiveOut

UEVar
a
VarKill
a
```

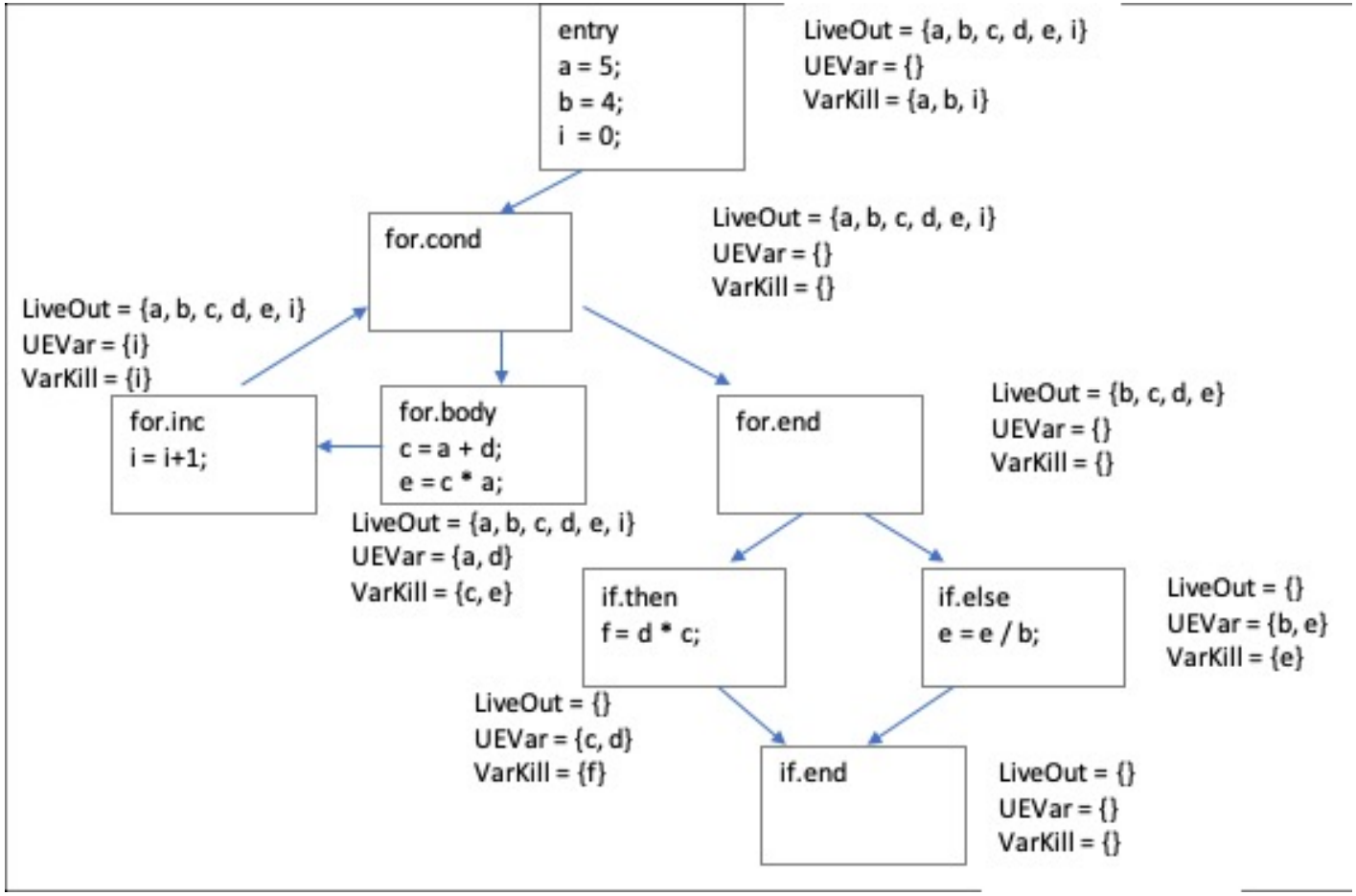


Lisa Chen
862186585
Project 3: Liveness Analysis

Test File 3.ll

```
Block name: entry
LiveOut
e b d c i a
UEVar

VarKill
a b i

Block name: for.cond
LiveOut
e b d c i a
UEVar

VarKill


Block name: for.body
LiveOut
e b d c a i
UEVar
a d
VarKill
c e

Block name: for.inc
LiveOut
e b d c i a
UEVar
i
VarKill
i
```

```
Block name: for.end
LiveOut
e b d c
UEVar

VarKill


Block name: if.then
LiveOut

UEVar
d c
VarKill
f

Block name: if.else
LiveOut

UEVar
e b
VarKill
e

Block name: if.end
LiveOut

UEVar

VarKill
```



Lisa Chen
862186585
Project 3: Liveness Analysis

Test File 4.ll

```
Block name: entry
LiveOut
i e a b
UEVar

VarKill
b a c e d i

Block name: for.cond
LiveOut
i e a b
UEVar

VarKill


Block name: for.body
LiveOut
c a i e
UEVar
a b
VarKill
c

Block name: while.cond
LiveOut
i c a e
UEVar
i e
VarKill
```

```
Block name: while.body
LiveOut
c a i e
UEVar
a i
VarKill
i

Block name: while.end
LiveOut
e a b i
UEVar
c a
VarKill
b

Block name: for.inc
LiveOut
i e a b
UEVar
i
VarKill
i

Block name: for.end
LiveOut

UEVar

VarKill
```