CS201 Compiler Construction

Project 3

Paraskevi Dimoragka {pdimo001}

USAGE:

```
$ opt -instnamer -load ../Pass/build/libLLVMLivenessAnalysisPass.so
-LivenessAnalysis < 1.ll> /dev/null
```

Implementation

The following 3 C++ Map containers hold the information for the **LIVEOUT**, **UEVAR** and **KILL** sets for each basic block of the function on which the **Liveness Analysis Pass** (LAP) is exercised:

```
std::map<const BasicBlock *, std::set<const AllocaInst *>> bb_live_out_map; // LIVEOUT
std::map<const BasicBlock *, std::set<const AllocaInst *>> bb_uevar_map; //  UEVAR
std::map<const BasicBlock *, std::set<const AllocaInst *>> bb_kill_map;  //  KILL
```

STEP-1

The first part of LAP performs **STEP 1** (as shown in the course slides) which is a forward direction pass of the basic blocks of the function's CFG, and computes the UEVAR and KILL variables for each basic block.

The following snippet of code calculates the UEVAR and KILL variables for the mathematical binary operators supported (i.e. +, -, *, /):

```
if (inst.isBinaryOp()) {

   const auto inst_op_code = inst.getOpcode();

   // Supported mathematical binary operations
   if (inst_op_code == Instruction::Add || inst_op_code == Instruction::Mul ||
       inst_op_code == Instruction::UDiv || inst_op_code == Instruction::SDiv ||
       inst_op_code == Instruction::Sub) {

       errs() << " , Op code: " << inst.getOpcodeName() << ")";

       const auto alloca_insts = getAllocaInstFromBinaryOp(inst);
       if (alloca_insts.first) {
           auto &bb_uevar = bb_uevar_map[&basic_block];
           // Check if it is already killed and if not add it to UEVAR
           const auto bb_kill = bb_kill_map[&basic_block];
           const auto bb_kill_it = bb_kill.find(alloca_insts.first);
```

```
            if (bb_kill_it == bb_kill.end()) {
                bb_uevar.insert(alloca_insts.first);
            }
        }
        if (alloca_insts.second) {
            auto &bb_uevar = bb_uevar_map[&basic_block];
            // Check if it is already killed and if not add it to UEVAR
            const auto bb_kill = bb_kill_map[&basic_block];
            const auto bb_kill_it = bb_kill.find(alloca_insts.second);
            if (bb_kill_it == bb_kill.end()) {
                bb_uevar.insert(alloca_insts.second);
            }
        }
    }
}
```

The following snippet calculates the UEVAR and KILL for the **StoreInst** and **ICmpInst** instruction:

```
if (const auto store_inst = dyn_cast_or_null<StoreInst>(&inst)) {
   //
   if (const auto alloca_value =
dyn_cast_or_null<AllocaInst>(store_inst->getOperand(1))) {
       auto &bb_kill = bb_kill_map[&basic_block];
       bb_kill.insert(alloca_value);
   }
   //
   if (const auto load_value = dyn_cast_or_null<LoadInst>(store_inst->getOperand(0)))
{
       const AllocaInst *alloca_inst =
dyn_cast_or_null<AllocaInst>(load_value->getOperand(0));
       auto &bb_uevar = bb_uevar_map[&basic_block];
       bb_uevar.insert(alloca_inst);
   }
}

if (const auto icm_inst = dyn_cast_or_null<ICmpInst>(&inst)) {
   //
   if (const auto load_value = dyn_cast_or_null<LoadInst>(icm_inst->getOperand(1))) {
       const AllocaInst *alloca_inst =
dyn_cast_or_null<AllocaInst>(load_value->getOperand(0));
       auto &bb_uevar = bb_uevar_map[&basic_block];
       bb_uevar.insert(alloca_inst);
   }
   //
   if (const auto load_value = dyn_cast_or_null<LoadInst>(icm_inst->getOperand(0))) {
       const AllocaInst *alloca_inst =
dyn_cast_or_null<AllocaInst>(load_value->getOperand(0));
       auto &bb_uevar = bb_uevar_map[&basic_block];
       bb_uevar.insert(alloca_inst);
   }
}
```

STEP-2

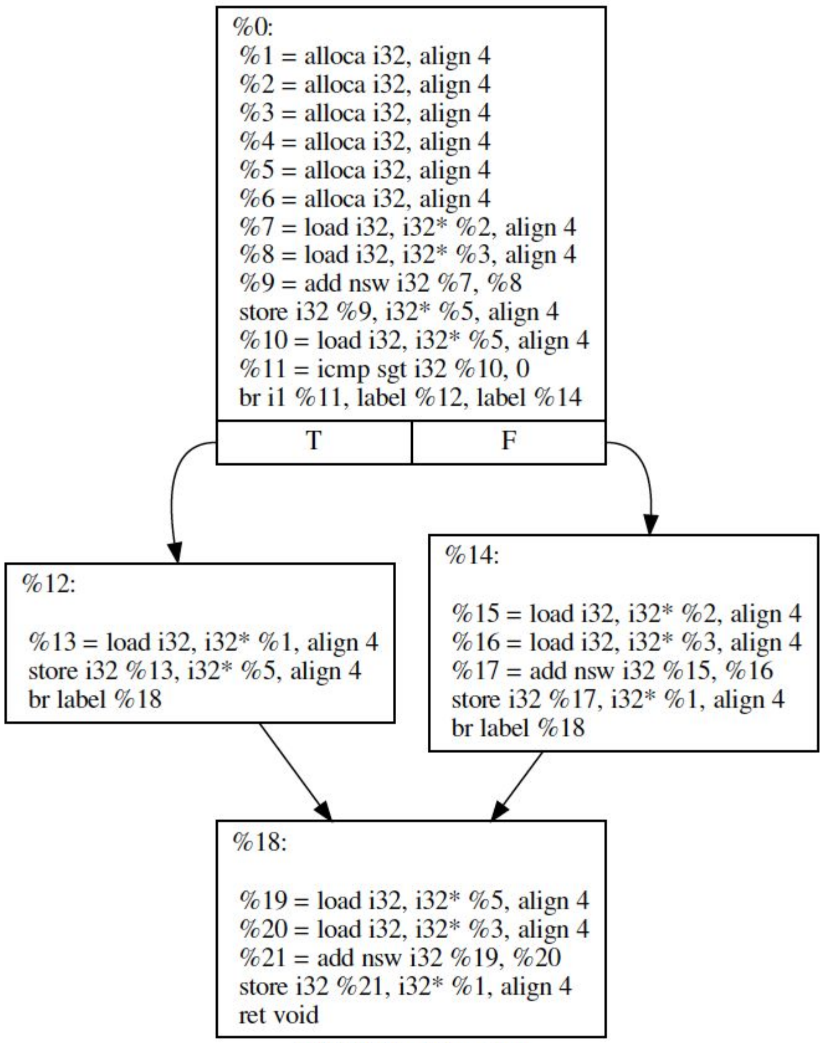
The second part of the LAP performs the **Basic Iterative Analysis** (as shown on the course slides) and computes the LIVEOUT set for each basic block of the function's CFG until convergence (i.e. all LIVEOUT sets remain unchanged):

```
//
// STEP 2: Basic Iterative Algorithm.
//
const auto &bb_list = F.getBasicBlockList();
bool test = true;
while(test) {
   test = false;
   for (auto bb_it = bb_list.rbegin(); bb_it != bb_list.rend(); ++bb_it) {
       auto previous_live_out = bb_live_out_map[&(*bb_it)];
       bb_live_out_map[&(*bb_it)].clear();
       auto & new_bb_live_out = bb_live_out_map[&(*bb_it)];
       for (const BasicBlock *succ : successors(&(*bb_it))) {
           const auto succ_bb_live_out = bb_live_out_map[succ];
           const auto succ_bb_kill = bb_kill_map[succ];
           const auto succ_bb_uevar = bb_uevar_map[succ];
           std::set<const AllocaInst*> diff;
           std::set_difference(succ_bb_live_out.begin(), succ_bb_live_out.end(),
                   succ_bb_kill.begin(), succ_bb_kill.end(), std::inserter(diff,
diff.begin()));
           std::set_union(diff.begin(), diff.end(), succ_bb_uevar.begin(),
succ_bb_uevar.end(),
                   std::inserter(new_bb_live_out, new_bb_live_out.begin()));
       }
       std::set<const AllocaInst*> intersect;
       std::set_difference(new_bb_live_out.begin(), new_bb_live_out.end(),
                             previous_live_out.begin(), previous_live_out.end(),
std::inserter(intersect, intersect.begin()));

       if (!intersect.empty()) {
           test = true;
       }
   }
}
```

Results on Test cases:

```
$ opt -instnamer -load ../Pass/build/libLLVMLivenessAnalysisPass.so
-LivenessAnalysis < 1.ll> /dev/null
```

Test case 1:



```
%0:
 %1 = alloca i32, align 4
 %2 = alloca i32, align 4
 %3 = alloca i32, align 4
 %4 = alloca i32, align 4
 %5 = alloca i32, align 4
 %6 = alloca i32, align 4
 %7 = load i32, i32* %2, align 4
 %8 = load i32, i32* %3, align 4
 %9 = add nsw i32 %7, %8
 store i32 %9, i32* %5, align 4
 %10 = load i32, i32* %5, align 4
 %11 = icmp sgt i32 %10, 0
 br i1 %11, label %12, label %14
```

```
%12:

 %13 = load i32, i32* %1, align 4
 store i32 %13, i32* %5, align 4
 br label %18
```

```
%14:

 %15 = load i32, i32* %2, align 4
 %16 = load i32, i32* %3, align 4
 %17 = add nsw i32 %15, %16
 store i32 %17, i32* %1, align 4
 br label %18
```

```
%18:

 %19 = load i32, i32* %5, align 4
 %20 = load i32, i32* %3, align 4
 %21 = add nsw i32 %19, %20
 store i32 %21, i32* %1, align 4
 ret void
```

CFG for 'test' function

<u>A) Terminal output:</u>

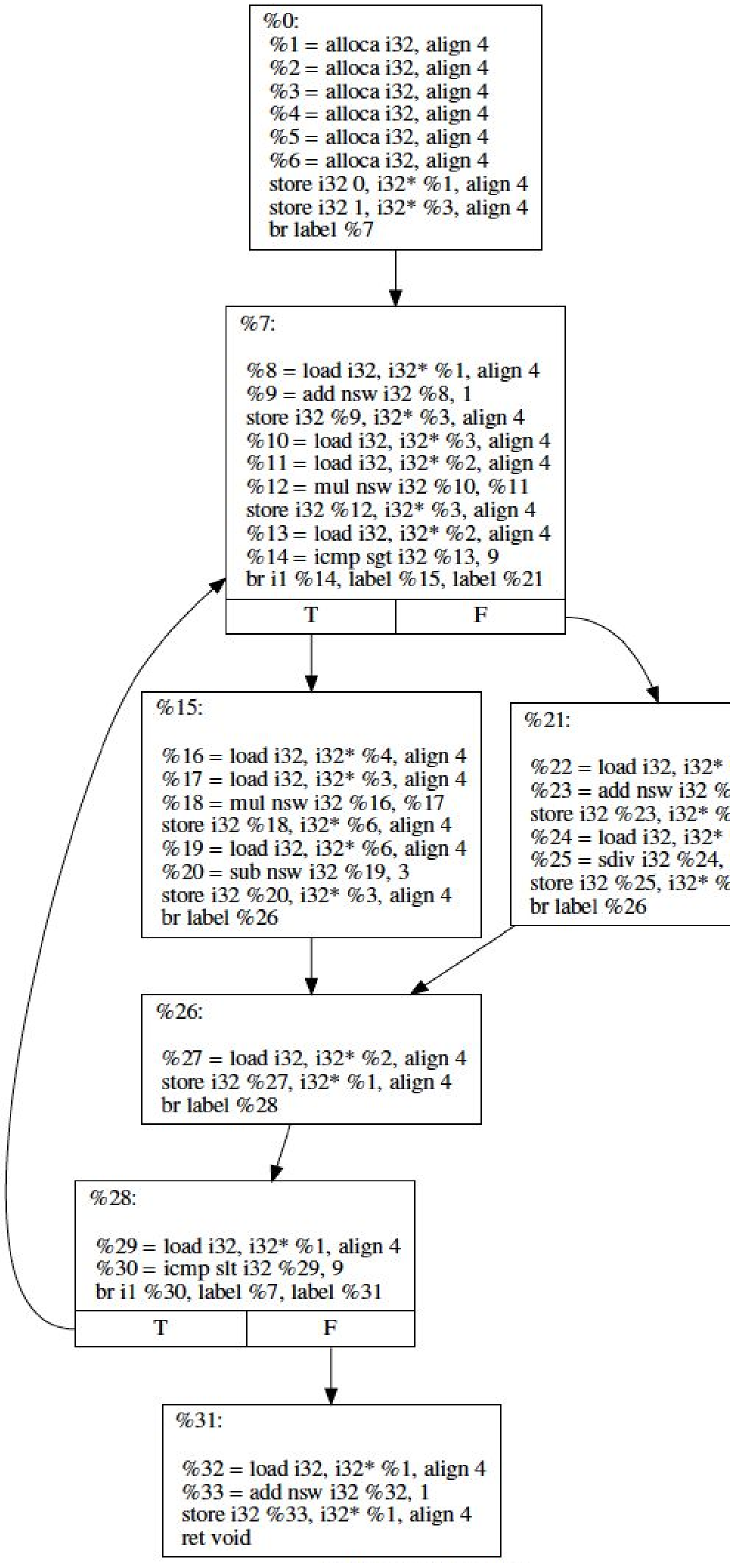
```
LivenessAnalysisPass: test

Basic block name: bb
-----------------
  %tmp = alloca i32, align 4 (number of Operands: 1
  %tmp1 = alloca i32, align 4 (number of Operands: 1
  %tmp2 = alloca i32, align 4 (number of Operands: 1
  %tmp3 = alloca i32, align 4 (number of Operands: 1
  %tmp4 = alloca i32, align 4 (number of Operands: 1
  %tmp5 = alloca i32, align 4 (number of Operands: 1
  %tmp6 = load i32, i32* %tmp1, align 4 (number of Operands: 1
  %tmp7 = load i32, i32* %tmp2, align 4 (number of Operands: 1
  %tmp8 = add nsw i32 %tmp6, %tmp7 (number of Operands: 2 , Op code: add)
  store i32 %tmp8, i32* %tmp4, align 4 (number of Operands: 2
  %tmp9 = load i32, i32* %tmp4, align 4 (number of Operands: 1
  %tmp10 = icmp sgt i32 %tmp9, 0 (number of Operands: 2
  br i1 %tmp10, label %bb11, label %bb13 (number of Operands: 3

Basic block name: bb11
-----------------
  %tmp12 = load i32, i32* %tmp, align 4 (number of Operands: 1
  store i32 %tmp12, i32* %tmp4, align 4 (number of Operands: 2
  br label %bb17 (number of Operands: 1

Basic block name: bb13
-----------------
  %tmp14 = load i32, i32* %tmp1, align 4 (number of Operands: 1
  %tmp15 = load i32, i32* %tmp2, align 4 (number of Operands: 1
  %tmp16 = add nsw i32 %tmp14, %tmp15 (number of Operands: 2 , Op code: add)
  store i32 %tmp16, i32* %tmp, align 4 (number of Operands: 2
  br label %bb17 (number of Operands: 1

Basic block name: bb17
-----------------
  %tmp18 = load i32, i32* %tmp4, align 4 (number of Operands: 1
  %tmp19 = load i32, i32* %tmp2, align 4 (number of Operands: 1
  %tmp20 = add nsw i32 %tmp18, %tmp19 (number of Operands: 2 , Op code: add)
  store i32 %tmp20, i32* %tmp, align 4 (number of Operands: 2
  ret void (number of Operands: 0

Liveness Analysis Pass (LAP)
==============================

Basic block name: bb
-----------------
UEVAR: tmp1 tmp2 tmp4
KILL: tmp4
LIVEOUT: tmp tmp1 tmp2 tmp4

Basic block name: bb11
-----------------
UEVAR: tmp
KILL: tmp4
```

```
LIVEOUT: tmp2 tmp4

Basic block name: bb13
-----------------
UEVAR: tmp1 tmp2
KILL: tmp
LIVEOUT: tmp2 tmp4

Basic block name: bb17
-----------------
UEVAR: tmp2 tmp4
KILL: tmp
LIVEOUT:
```

B) Contents of 1.out program generated file:

```
Liveness Analysis Pass (LAP)
===============================

Basic block name: bb
-----------------
UEVAR: tmp1 tmp2 tmp4
KILL: tmp4
LIVEOUT: tmp tmp1 tmp2 tmp4

Basic block name: bb11
-----------------
UEVAR: tmp
KILL: tmp4
LIVEOUT: tmp2 tmp4

Basic block name: bb13
-----------------
UEVAR: tmp1 tmp2
KILL: tmp
LIVEOUT: tmp2 tmp4

Basic block name: bb17
-----------------
UEVAR: tmp2 tmp4
KILL: tmp
LIVEOUT:
```

Test case 2 with back edge:



```
%0:
 %1 = alloca i32, align 4
 %2 = alloca i32, align 4
 %3 = alloca i32, align 4
 %4 = alloca i32, align 4
 %5 = alloca i32, align 4
 %6 = alloca i32, align 4
 store i32 0, i32* %1, align 4
 store i32 1, i32* %3, align 4
 br label %7
```

```
%7:

 %8 = load i32, i32* %1, align 4
 %9 = add nsw i32 %8, 1
 store i32 %9, i32* %3, align 4
 %10 = load i32, i32* %3, align 4
 %11 = load i32, i32* %2, align 4
 %12 = mul nsw i32 %10, %11
 store i32 %12, i32* %3, align 4
 %13 = load i32, i32* %2, align 4
 %14 = icmp sgt i32 %13, 9
 br i1 %14, label %15, label %21
```

```
%15:

 %16 = load i32, i32* %4, align 4
 %17 = load i32, i32* %3, align 4
 %18 = mul nsw i32 %16, %17
 store i32 %18, i32* %6, align 4
 %19 = load i32, i32* %6, align 4
 %20 = sub nsw i32 %19, 3
 store i32 %20, i32* %3, align 4
 br label %26
```

```
%21:

 %22 = load i32, i32*
 %23 = add nsw i32 %
 store i32 %23, i32* %
 %24 = load i32, i32*
 %25 = sdiv i32 %24,
 store i32 %25, i32* %
 br label %26
```

```
%26:

 %27 = load i32, i32* %2, align 4
 store i32 %27, i32* %1, align 4
 br label %28
```

```
%28:

 %29 = load i32, i32* %1, align 4
 %30 = icmp slt i32 %29, 9
 br i1 %30, label %7, label %31
```

```
%31:

 %32 = load i32, i32* %1, align 4
 %33 = add nsw i32 %32, 1
 store i32 %33, i32* %1, align 4
 ret void
```

CFG for 'test' function

```
$ opt -instnamer -load ../Pass/build/libLLVMLivenessAnalysisPass.so
-LivenessAnalysis < 2.ll> /dev/null
```

A) Terminal output:

```
LivenessAnalysisPass: test

Basic block name: bb
-----------------
  %tmp = alloca i32, align 4 (number of Operands: 1
  %tmp1 = alloca i32, align 4 (number of Operands: 1
  %tmp2 = alloca i32, align 4 (number of Operands: 1
  %tmp3 = alloca i32, align 4 (number of Operands: 1
  %tmp4 = alloca i32, align 4 (number of Operands: 1
  %tmp5 = alloca i32, align 4 (number of Operands: 1
  store i32 0, i32* %tmp, align 4 (number of Operands: 2
  store i32 1, i32* %tmp2, align 4 (number of Operands: 2
  br label %bb6 (number of Operands: 1

Basic block name: bb6
-----------------
  %tmp7 = load i32, i32* %tmp, align 4 (number of Operands: 1
  %tmp8 = add nsw i32 %tmp7, 1 (number of Operands: 2 , Op code: add)
  store i32 %tmp8, i32* %tmp2, align 4 (number of Operands: 2
  %tmp9 = load i32, i32* %tmp2, align 4 (number of Operands: 1
  %tmp10 = load i32, i32* %tmp1, align 4 (number of Operands: 1
  %tmp11 = mul nsw i32 %tmp9, %tmp10 (number of Operands: 2 , Op code: mul)
  store i32 %tmp11, i32* %tmp2, align 4 (number of Operands: 2
  %tmp12 = load i32, i32* %tmp1, align 4 (number of Operands: 1
  %tmp13 = icmp sgt i32 %tmp12, 9 (number of Operands: 2
  br i1 %tmp13, label %bb14, label %bb20 (number of Operands: 3

Basic block name: bb14
-----------------
  %tmp15 = load i32, i32* %tmp3, align 4 (number of Operands: 1
  %tmp16 = load i32, i32* %tmp2, align 4 (number of Operands: 1
  %tmp17 = mul nsw i32 %tmp15, %tmp16 (number of Operands: 2 , Op code: mul)
  store i32 %tmp17, i32* %tmp5, align 4 (number of Operands: 2
  %tmp18 = load i32, i32* %tmp5, align 4 (number of Operands: 1
  %tmp19 = sub nsw i32 %tmp18, 3 (number of Operands: 2 , Op code: sub)
  store i32 %tmp19, i32* %tmp2, align 4 (number of Operands: 2
  br label %bb25 (number of Operands: 1
Basic block name: bb20
-----------------
  %tmp21 = load i32, i32* %tmp4, align 4 (number of Operands: 1
  %tmp22 = add nsw i32 %tmp21, 1 (number of Operands: 2 , Op code: add)
  store i32 %tmp22, i32* %tmp, align 4 (number of Operands: 2
  %tmp23 = load i32, i32* %tmp3, align 4 (number of Operands: 1
  %tmp24 = sdiv i32 %tmp23, 2 (number of Operands: 2 , Op code: sdiv)
  store i32 %tmp24, i32* %tmp4, align 4 (number of Operands: 2
  br label %bb25 (number of Operands: 1

Basic block name: bb25
-----------------
  %tmp26 = load i32, i32* %tmp1, align 4 (number of Operands: 1
  store i32 %tmp26, i32* %tmp, align 4 (number of Operands: 2
  br label %bb27 (number of Operands: 1

Basic block name: bb27
-----------------
  %tmp28 = load i32, i32* %tmp, align 4 (number of Operands: 1
  %tmp29 = icmp slt i32 %tmp28, 9 (number of Operands: 2
  br i1 %tmp29, label %bb6, label %bb30 (number of Operands: 3

Basic block name: bb30
-----------------
  %tmp31 = load i32, i32* %tmp, align 4 (number of Operands: 1
  %tmp32 = add nsw i32 %tmp31, 1 (number of Operands: 2 , Op code: add)
  store i32 %tmp32, i32* %tmp, align 4 (number of Operands: 2
  ret void (number of Operands: 0

Liveness Analysis Pass (LAP)
==============================

Basic block name: bb
-----------------
UEVAR:
KILL: tmp tmp2
LIVEOUT: tmp tmp1 tmp3 tmp4

Basic block name: bb6
-----------------
UEVAR: tmp tmp1
KILL: tmp2
LIVEOUT: tmp1 tmp2 tmp3 tmp4
```

```
Basic block name: bb14
-----------------
UEVAR: tmp2 tmp3
KILL: tmp2 tmp5
LIVEOUT: tmp1 tmp3 tmp4

Basic block name: bb20
-----------------
UEVAR: tmp3 tmp4
KILL: tmp tmp4
LIVEOUT: tmp1 tmp3 tmp4

Basic block name: bb25
-----------------
UEVAR: tmp1
KILL: tmp
LIVEOUT: tmp tmp1 tmp3 tmp4

Basic block name: bb27
-----------------
UEVAR: tmp
KILL:
LIVEOUT: tmp tmp1 tmp3 tmp4

Basic block name: bb30
-----------------
UEVAR: tmp
KILL: tmp
LIVEOUT:
```

B) Contents of 2.out program generated file:

```
Liveness Analysis Pass (LAP)
==============================

Basic block name: bb
-----------------
UEVAR:
KILL: tmp tmp2
LIVEOUT: tmp tmp1 tmp3 tmp4

Basic block name: bb6
```

```
-----------------
UEVAR: tmp tmp1
KILL: tmp2
LIVEOUT: tmp1 tmp2 tmp3 tmp4

Basic block name: bb14
-----------------
UEVAR: tmp2 tmp3
KILL: tmp2 tmp5
LIVEOUT: tmp1 tmp3 tmp4

Basic block name: bb20
-----------------
UEVAR: tmp3 tmp4
KILL: tmp tmp4
LIVEOUT: tmp1 tmp3 tmp4

Basic block name: bb25
-----------------
UEVAR: tmp1
KILL: tmp
LIVEOUT: tmp tmp1 tmp3 tmp4

Basic block name: bb27
-----------------
UEVAR: tmp
KILL:
LIVEOUT: tmp tmp1 tmp3 tmp4

Basic block name: bb30
-----------------
UEVAR: tmp
KILL: tmp
LIVEOUT:
```

Where: tmp, tmp1, tmp2, tmp3, tmp4, tmp5 is a, b, c, d, e, f respectively