```
// 15-745 S14 Assignment 2: liveness.cpp
// Group: aebtekar, auc
////////////////////////////////////////////////////////////////////////////////

#include "llvm/IR/Function.h"
#include "llvm/Pass.h"

#include "dataflow.h"

using namespace llvm;

namespace {

// 1-1 mapping between indices and variables
std::vector<std::string> itov;
std::map<Value*, int> vtoi;

Elem livenessTransition(Instruction* instr, Elem elem)
{
  // kill defined variable
  int idx = vtoi[instr] - 1;
  if (idx != -1)
  {
    elem[idx] = false;
  }
  // generate used variables
  for (User::op_iterator OI = instr->op_begin(), OE = instr->op_end(); OI != OE; ++
OI)
  {
    Value* val = *OI;
    if (isa<Instruction>(val) || isa<Argument>(val))
    {
      idx = vtoi[val] - 1;
      if (idx != -1)
      {
        elem[idx] = true;
      }
    }
  }
  return elem;
}

class Liveness : public FunctionPass {
 public:
  static char ID;

  Liveness() : FunctionPass(ID) { }

  virtual bool runOnFunction(Function& F) {
    //ExampleFunctionPrinter(errs(), F);

    itov.clear();
    vtoi.clear();
    // find variables passed as arguments
    for (ilist_iterator<Argument> AI = F.arg_begin(), AE = F.arg_end(); AI != AE; +
+AI)
    {
      std::string name = "%";
      name += AI->getName();
      itov.push_back(name);
      vtoi[AI] = itov.size();
    }
    // find variables declared by instructions
    for (ilist_iterator<BasicBlock> BI = F.begin(), BE = F.end(); BI != BE; ++BI)
    for (ilist_iterator<Instruction> II = BI->begin(), IE = BI->end(); II != IE; ++
II)
    {
      std::string name;
      raw_string_ostream stream(name);
      II->print(stream);
      // check if it's a variable definition
      size_t st = name.find('%');
      size_t fi = name.find('=');
      if (st < fi && fi != std::string::npos)
      {
        // if so, include its name in the lattice
        name = name.substr(st, fi-st-1);
        itov.push_back(name);
        vtoi[II] = itov.size();
      }
    }
    // define lattice and do the analysis
    Lattice lattice(itov, false);
    backwardSearch(F, &lattice, &livenessTransition);

    // Did not modify the incoming Function.
    return false;
  }

  virtual void getAnalysisUsage(AnalysisUsage& AU) const {
    AU.setPreservesCFG();
  }

 private:
};

char Liveness::ID = 0;
RegisterPass<Liveness> X("cd-liveness", "15745 Liveness");

}
```