Homework #9

Question 1 (7 pt.)

The goal of this assignment is writing an LLVM pass for a reaching definitions analysis. The pass should apply in a per-function basis (i.e., the pass should be a child of the `llvm::FunctionPass` class), and should print the *IN* and *OUT* sets of reaching definitions for each basic block as the final output. Consider the following hints:

- Since LLVM IR is in single static assignment (SSA) form, the *KILL* set of definitions is always empty. Also, since every variable can be defined only once by an instruction, the statement defining a variable will be referred to by the defined variable itself. Thus, our *IN* and *OUT* sets will be sets (`std::set`) of strings representing LLVM variable names.

- You can use the following two functions, operating on C++ sets of strings, to calculate the union of two sets, and print the content of a set. The function calculating the union will return whether the target set is actually modified by the operation or not. This condition is useful to determine the convergence of the algorithm.

```
bool ExtendSet(std::set<std::string> &set1, std::set<std::string> &set2)
{
        int last_size = set1.size();
        for (const std::string &s : set2)
                set1.insert(s);
        return set1.size() - last_size;
}

void PrintSet(std::set<std::string> &set)
{
        llvm::errs() << "{";
        for (const std::string &s : set)
                llvm::errs() << ' ' << s;
        llvm::errs() << " }";
}
```

- For each basic block found in the function, you need to attach metadata in the form of three sets: *GEN*, *IN*, and *OUT*. You can pack these three fields in the following data structure:

```
struct BasicBlockInfo
{
        std::set<std::string> gen;
        std::set<std::string> in;
        std::set<std::string> out;
};
```

An instance of this structure can then be associated to each basic block by using a hash table indexed by a basic block name, and containing elements of type `BasicBlockInfo*`, as follows:

```
std::unordered_map<std::string, BasicBlockInfo *> basic_block_table;
```

You can use the LLVM pass provided in the support material as a starting point. In this example, you can see how to traverse basic blocks, instructions, and arguments, as well as how to traverse the set of successors and predecessors for each basic block.

Upload your code in a file named `reaching-def.cc`. The file should compile and run without errors on the Linux COE machines using command

```
g++ -fPIC -shared \
         reaching-def.cc \
         -o reaching-def.so \
         -std=c++11 \
         `llvm-config --cppflags`
```

Question 2 (3 pt.)

Write a small function in C that produces a non-trivial control flow graph that requires at least two iterations for the reaching definitions analysis to converge. Provide the LLVM code for this C program as produced by our MiniC parser. Invoke the LLVM `opt` tool to run your analysis pass, and show the command line you used. Include the output of the pass, and comment on the results you observe. Verify that this is the expected output based on the LLVM code produced by the parser.

Upload the code, command lines, and text for this question in file `hw9.pdf`.