First Project: bytecodes to dot - 10 points

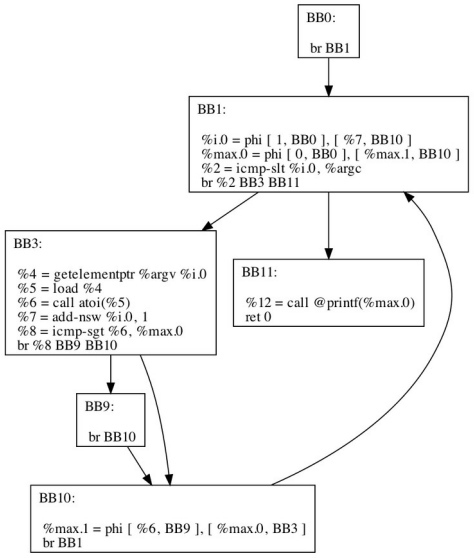
The goal of the first project is to write a LLVM pass that prints bytecode programs into the dot format. Notice that LLVM already have such a pass, which can be invoked with the command line: `opt -view-cfg file.bc`, but we, of course, will not use this tool. You must write your own pass, and it must have a few requirements:

- Each instruction must be printed with an opcode.
- All the arguments of an instruction must be printed next to this instruction.
- Arguments that do not have names, such as getelementptr in function calls, should not be printed.
- Type information should not be printed.

As an example, consider the program below:

```
int main(int argc, char** argv) {
  int i = 1;
  int max = 0;
  while (i < argc) {
    int aux = atoi(argv[i]);
    i++;
    if (aux > max) {
      max = aux;
    }
  }
  printf("Max = %d\n", max);
}
```

For this program, you must produce an output similar to the following figure (click to enlarge), which was constructed with the graphviz tool, publicly available to linux and OSX:



Notice that the exact shape of your CFG will depend on how you have produced the LLVM IR. It's ok if your output is not exactly the same as the example above, as long as the four requirements are observed. The figure above was produced from this DOT, which I have transcribed below. Notice that DOT files are printed in simple ASCII characters. If `dot` is installed in your system, you can produce that file with the syntax: `dot -Tpdf file.dot -o file.pdf`.

```
digraph "CFG for 'main' function" {
        AAA [shape=record,
             label="{BB0:\l\l  br BB1\l}"];
        AAA -> AAB;
        AAB [shape=record,
            label="{BB1:\l\l
                    %i.0 = phi [ 1, BB0 ], [ %7, BB10 ]\l
                    %max.0 = phi [ 0, BB0 ], [ %max.1, BB10 ]\l
                    %2 = icmp-slt %i.0, %argc\l
                    br %2 BB3 BB11\l}"];
        AAB -> AAC;
        AAB -> AAD;
        AAC [shape=record,
             label="{BB3:\l\l
                     %4 = getelementptr %argv %i.0\l
                     %5 = load %4\l
                     %6 = call atoi(%5)\l
                     %7 = add-nsw %i.0, 1\l
                     %8 = icmp-sgt %6, %max.0\l
                     br %8 BB9 BB10\l}"];
        AAC -> AAE;
        AAC -> AAF;
        AAE [shape=record,
             label="{BB9:\l\l  br BB10\l}"];
        AAE -> AAF;
        AAF [shape=record,
             label="{BB10:\l\l
             %max.1 = phi [ %6, BB9 ], [ %max.0, BB3 ]\l
```

```
            br BB1\l}"];
        AAF -> AAB;
        AAD [shape=record,
             label="{BB11:\l\l
             %12 = call @printf(%max.0)\l
             ret 0\l}"];
}
```

What must be turned in: each group must e-mail the instructor a zip file, containing the pass, plus a README.txt file. The zip file must contain the directory in which the pass was implemented, including the Makefile. The instructor must be able to compile the pass by either:

- Unpacking the directory in /llvm/lib/Transforms, and then typing make.
- Unpacking the directory in a folder, outside the LLVM tree, and then updating the Makefile
  (e.g., LLVM_SRC_ROOT and LLVM_OBJ_ROOT) to indicate where are the build and llvm root folders are located (and then typing make, of course).

The README.txt file must contain the command line that should be used to run the pass, and how the pass is compiled, if within, or outside the LLVM tree. Additionally, the zip file must contain a PDF for each CFG that the pass produces for the Stanford benchmarks.