## Assignment Description

In this assignment, you will implement several dataflow analyses and optimizations. We expect you to build upon the code that you wrote for the previous assignments. You will implement the following.

**1. Control-flow Graph Construction**.

You will first build a control-flow graph representation of the program, starting from the three-address code representation in the previous assignment. We suggest you use a simple representation where each CFG node is a single instruction.

**2. A Generic Dataflow Analysis Framework**.

Your implementation of the dataflow analysis framework will include a generic dataflow engine and an interface for the dataflow information and for the transfer functions. For each analysis instance, you only need to implement this interface; the analysis engine is implemented once and reused for each analysis instance.

The dataflow analysis class must have a method which, given a control flow graph and a boundary dataflow information, solves the dataflow equations using the worklist algorithm, and computes dataflow information at each program point in the flow graph. You should allow your dataflow analysis class to be instantiated either as a forward or as a backward analysis.

**3. Analysis Instances and Optimizations**.

Use your generic dataflow analysis framework to implement the following analysis instances:

- *Constant folding analysis*: determine variables whose values are constant.
- *Live variables analysis*: compute variables which may be live at each program point;
- *Available expressions*: compute expressions available in all of the program executions;

Next, use the analysis results to perform the following optimizations:

- *Constant folding*: use the results from constant folding analysis and replace each constant variable with the computed constant.
- *Dead code elimination*: remove code which updates variables whose values are not used in any executions. This optimization will use the results from the live variable analysis.
- *Common subexpression elimination*: reuse expressions already computed in the program. Here you will use the information computed with the available expressions analysis. If an expression is available before an instruction that re-computes that expression, you have to replace the computation by the variable which holds the result of that expression in all executions of the program. If there is no such variable, you will

create a temporary variable to store the result of the expression. Common subexpression elimination shoud also remove redundant run-time checks such as array bounds checks or null pointer checks.

**Command line invocation**. As in the previous assignment, your compiler must be invoked with a single file name as argument: `java IC.Compiler <file.ic>`. With this command, the compiler will parse the input file, perform semantic checks, generate intermediate code, and build the control flow graph. Then, it will perform various analyses and optimizations on the three-address code. In addition to all of the options from the previous assignments, your compiler must support the following command-line options:

- Option `-cfo` for constant folding;
- Option `-dce` for dead code elimination;
- Option `-cse` for common subexpression elimination;
- Option `-opt` to perform all of the above optimizations.

The compiler will perform the optimizations in the order they occur in the command line. The above arguments may be invoked multiple times in the same command line - in that case the compiler will execute them multiple times, in the specified order. The compiler will only perform the analyses necessary for the optimizations requested. When the `-print-ir` also occur in the command line, you must print the Low IR before or after optimizations, depending on where it occurs in the command line. For instance, with: `-cfo -print-ir -dce`, you must print the Low IR after the compiler performs constant folding, but before it removes dead code.

**Output dataflow information** Your compiler will also print out the computed dataflow information at selected program points. With the command line option `-print-dfa`, the compiler must print the dataflow information at each point in the program. Make sure your output is readable. Each dataflow fact must clearly indicate the program statement that it refers to, and whether it represent the information before or after the statement.

### Bonus Points

For an additional 10 bonus points, implement PRE, partial redundancy elimination, as discussed in class. Mention in the writeup if you have implemented this transformation. The command line option for this transformation is `-pre`.

## What to turn in

- A file `pa4.tar.gz` containing all of your source code and test cases (in directories `/src` and `/test`).
- A brief, clear, and concise document describing the your code structure and testing strategy.