HW Assignment 3

Total = 60

due date: 8th December

1. **Mark the correct answer. More than one answers can be correct: 2*5 = 10**
   a. A typical dataflow analysis
      i. is computed over control flow graphs
      ii. may not reach fixed-point.
      iii. is linear in runtime in the number of instructions in the compiled program.

   b. Abstract Syntax Tree of a source code:
      i. includes all the source code elements including punctuation and delimiters.
      ii. can be enhanced with information such as properties and annotations for every element it contains.
      iii. usually contains information about the position of an element in the source code.

   c. Along a control flow graph, a node d **strictly dominates** node i,
      i. if all paths from entry to node i include d but d != i
      ii. if all paths from entry to node i include d
      iii. if every possible path from i to exit includes d

   d. A reaching definition is:
      i. May Forward analysis
      ii. Must Forward analysis
      iii. May Backward analysis
      iv. Must Backward analysis

   e. A definition d is reached at a particular program point n
      i. If d is killed at node n-1
      ii. If d is killed at node n
      iii. If d is killed at node n+1

Problem 2. Consider the following code snippet where variables a, b, c and d are integers.

```
1              a :=  2*a - b
2              b :=  c - a
3      L0:     c :=  c - b
4               if c  > a goto L1
5              b :=  b * a
6              d :=  b - a
7               if d  < 2 goto L2
8              else  goto L0
9      L1:     a :=  b - c
10             c :=  d + b
11              if c  > -3 goto L2
12             else  goto L1
13     L2:     a :=  0
14             b :=  2*b
15 return
```

a. Identify the basic blocks in this piece of code, explicitly writing out the line numbers that belong to each block. For example, if you think that lines 42 and 43 constitute a basic block, and lines 100, 101, 102 make another basic block, you should write both blocks as: {42,43}, {100,101,102}. **5**

b. Draw the control flow graph (CFG) of this example, where each node is a statement. **5**

c. Recall that given two nodes v and u in a CFG, v dominates u if all paths from entry to u include v. For each node u in your CFG, identify all the other nodes v that dominate u. **5**

d. Recall that a back-edge is an edge in the CFG such that the target of the edge dominates the source. Identify all the back-edges in your CFG, clearly stating the source and target of each edge you identify. **5**

Problem 3. Data flow analysis derives information about the behavior of a program by examining the code statically. In the lecture slides, there is an example of liveness analysis of variables, and how we can solve data flow equations to compute the liveness of variables.

Now you need to do the same for **available expressions**. An expression, **x+y**, is **available** at node n if every path from the entry node to n evaluates **x+y**, and there are no definitions of **x** or **y** after the last evaluation.

Now, considering the following code:

```
x := a + b;
y := a * b;
z := a - b;
 while (z > b) {
   while (y > a) {
     a := a + 1;
     x := a + b;
   }
  z:= a - b;
}
```

a) Draw the CFG for this code snippet 4

b) What is the dataflow equation of **available expressions**? 6

Hint: Use the following notations:

i. IN - set of expressions available at start of block

ii. OUT - set of expressions available at end of block

iii. GEN - set of expressions computed in block

iv. KILL - set of expressions killed in in block

c) Work through the algorithm for computing available expression by drawing a table like the one shown above. Solve the dataflow equations and compute the final IN and OUT for each node. 10

Problem 4: Consider the following example:

```
#define MAX (1024*1024*32)
#define REP  100
#define  B    (16*1024)

int main() {
     int i,j,r;
     char array[MAX];

     clock_t t1 = clock();

     // Tiled loop
     for (i = 0; i < MAX; i += B)  {
          for (r = 0; r < REP; r++) {
                for (j = i; j < (i + B); j+=64) {
                     array[j] = r;
                }
          }
     }
     clock_t t2 = clock();

     // un-tiled loop
     for (r = 0; r < REP; r++) {
          for (i = 0; i < MAX; i+=64)  {
                array[i] = r;
          }
     }

     clock_t t3 = clock();

     tiled_time = (double)(t2 - t1) /  CLOCKS_PER_SEC);
     untiled_time = (double)(t3 - t2) /  CLOCKS_PER_SEC);

}
```

Between `tiled_time` and `untiled_time`, which one will take more time and why? **10**