Tutorial 5 (Optimisation)
(to be covered in 1.5 tutorials)

1. Consider the Jimple-like code snippet shown in Figure Q1. Draw the corresponding Control Flow Graph. Which variables are *live* and which are *dead after* the node for the instruction at label `l1`? Explain carefully the reasons for your answer.

```
    x = 1;
    w = 0;
    z = 3;
l0: if z==0 goto l4;
l1: y = w;
    if z==3 goto l2;
    y = x-1;
    w = y+z;
    goto l3;
l2: x = z;
    y = x+1;
l3: z = z-1;
    goto l0;
l4: return ;
```

Figure Q1

2. The transfer equations for *liveness analysis* for a Control Flow Graph with 12 nodes are as shown in Figure Q2. For simplicity, the $out_L(n)$ expressions have been substituted away. Solve the equations *iteratively* by filling in successive columns of Table Q2 until there are no further changes.

$in_L(1) = in_L(2)$
$in_L(2) = in_L(3) \setminus \{z\}$
$in_L(3) = in_L(4) \setminus \{x\}$
$in_L(4) = in_L(11) \cup in_L(5) \cup \{x, z\}$
$in_L(5) = in_L(6) \setminus \{r\} \cup \{x\}$
$in_L(6) = in_L(7) \setminus \{t\} \cup \{z\}$
$in_L(7) = in_L(8) \cup in_L(9) \cup \{t, z\}$
$in_L(8) = in_L(10) \setminus \{x\} \cup \{r, t\}$
$in_L(9) = in_L(10) \setminus \{x\} \cup \{r, z\}$
$in_L(10) = in_L(4) \cup \{z\}$
$in_L(11) = in_L(12) \cup \{r\}$
$in_L(12) = \varnothing$

Figure Q2

Table Q2

	0	1	2	…
$in_L(1)$	∅			
$in_L(2)$	∅			
$in_L(3)$	∅			
$in_L(4)$	∅			
$in_L(5)$	∅			
$in_L(6)$	∅			
$in_L(7)$	∅			
$in_L(8)$	∅			
$in_L(9)$	∅			
$in_L(10)$	∅			
$in_L(11)$	∅			
$in_L(12)$	∅			

3. Consider the Control Flow Graph (CFG) shown in Figure Q3. Derive the set of transfer functions for *available expressions* analysis, i.e. the functions for $in_A(n)$ and $out_A(n)$ for each node n in the CFG.

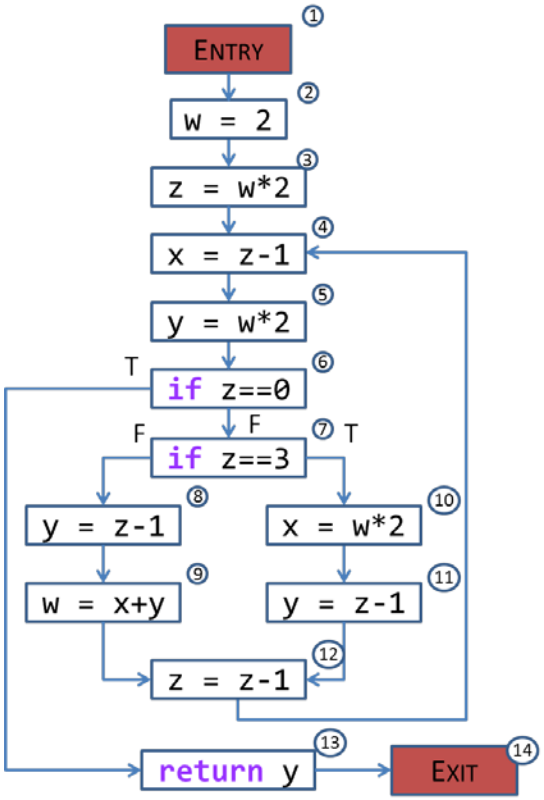

Figure Q3

4. Use the *worklist* algorithm shown in Figure Q4 to solve the set of transfer functions derived in Question 3. You may assume that the worklist is initialized to the set of all nodes, in increasing order of node number, as shown in Figure Q3.

```
worklist = [ all nodes ]
while worklist != empty do
   m = removeFirst(worklist)
   recompute out_A(m)
   if out_A(m) has changed then
      for each successor n of m
         compute in_A(n)
         if in_A(n) has changed then put n into worklist
                                (if not already in worklist)
```

Figure Q4