NAME: **ROLL NO:**

CS618: Program Analysis
Mid Semester Examination, 2016-17 I

Max Time: 2 Hours Max Marks: 125

NOTE:

- There are total **5** questions on **4 pages**
- Write your name and roll number on the question paper and the answer book.
- No explanations will be provided. In case of a doubt, make suitable assumptions and justify.
- Presenting your answers properly is your responsibility. You lose credit if you can not present your ideas clearly, and in proper form. Please DO NOT come back for re-evaluation saying, "What I actually meant was . . . ".
- Be precise and write clearly. Remember that somebody has to read it to evaluate!

---

1. Consider a flow graph G with a unique entry node ENTRY that dominates all nodes of G. Prove that every node in G except ENTRY has a unique *immediate dominator*. By definition, immediate dominator of a node n is the *closest strict dominator* of n. [10]

   - Every node except ENTRY has at least one strict dominator. Consider a node Z that has more than one strict dominators. Consider two such dominators X and Y. Then, it can be proved that either X dominates Y or Y dominates X (*proof below). Thus, it possible to find *least* element among all strict dominators of Z. This element is the desired immediate dominator.
   - * Consider a cycle free path from ENTRY→Z. Because both X and Y strictly dominate Z, they must occur on this path. WLOG, assume that X occurs before Y in this path. Thus the path is: Entry → X → Y → Z. We will prove that X must dominate Y.

     Assume the contrary (X does not dominate Y). Then, we have a path Entry $\Rightarrow Y$ free of X. But then, Entry ⇒ Y → Z is a path to Z free of X. But that also contradicts the fact that X dominates Z.

2. The original definition of *Dominance Frontier* (df) is: A node m is in df(n) if

   (a) n dominates a predecessor of m in the flow graph, and

   (b) n does not strictly dominate m

   Dr Dominoz thinks the following modified definition of df(n) is equivalent as far as computation of SSA form is concerned: A node m is in df(n) if

   (a) n dominates a predecessor of m in the flow graph, and

(b) n does not dominate m

i.e., Dr Dominoz has dropped the term "strictly" from the definition.

You jobs is to either **prove that Dr Dominoz is right** or show that he is **wrong, by giving a counter example**. The proof must work for any arbitrary CFG, while the counter example must show an incorrect SSA form being generated for a CFG. [10]

3. Consider the following extensions to our 3-address code language:

   - `x.lock`: "locks" a variable **x**
   - `x.unlock`: "unlocks" a variable **x**
   - `x.secureComp` does some "secure computation" on **x**

   In this language, a "secure computation" on a variable **x** is allowed only when it is locked (**x**.lock executed before **x**.secureComp, without an intervening **x**.unlock). We call such secureComp **safe**, otherwise it is **unsafe**.

   The language obviously contains basic constructs like *assignment* statements, *goto* statements, and conditionals (*if-goto*). Following semantic properties hold:

   - All variables are **unlocked** at the *entry*
   - **lock** and **unlock** operations are idempotent (Locking a locked variable is allowed, but it has no effect on the lock-status of the variable. Similarly, unlocking an unlocked variable is also allowed)

   Here are couple of sample programs, `PROGRAM-1` is valid and `PROGRAM-2` is invalid.

```
// PROGRAM-1
  c = 5
  n = 0
  t = c > 0
  if t goto L1
  n.lock
  n.secureComp
  goto L2

L1:
  c.lock
  c.secureComp
  d = e - 5;
  c.unlock
L2:
  n.lock

  n.secureComp

  n.unlock
  c.unlock
```

```
// PROGRAM-2
  c = 5
  n = 0
  t = c > 0
  if t goto L1
    n.lock
    c.secureComp // BAD, c not locked
    c.unlock
    goto L2
  L1:
    n.lock
    n.secureComp
    d  = e - 5;
    c.lock
  L2:
    n.secureComp // OK, n locked on
                 // all paths
    c.secureComp // BAD,  c may not
                 // be locked
    n.unlock
    c.unlock
```

   Design an **intraprocedural** data flow analysis framework to mark unsafe secure computations ("secureComp"). In particular,

   (a) Draw the lattice for the framework, and describe it briefly. [5]

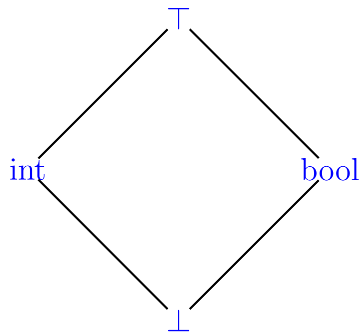
   (b) Describe the meet operator ($\wedge$). [5]

(c) Describe intuitively the meaning of the top and the bottom elements. [5]

(d) Define the flow functions for statements. You do not need to list all types of statements, but use suitable representatives (for e.g. `x op y` to represent binary operators). [10]

(e) Is your framework *Forward* or *Backward*? Justify your answer. Also describe the *BoundaryInfo* (initialization information at the boundary of the flow graph). [2+3]

(a) Lattice:



(b) meet:

$\wedge$	$\top$	int	bool	$\bot$
$\top$	$\top$	int	bool	$\bot$
int	int	int	$\bot$	$\bot$
bool	bool	$\bot$	bool	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

(c) $\top$ represents no type inferred yet. $\bot$ represents conflicting types inferred (more than one type for a variable).

(d) Flow functions. (This is one of the many possible solutions)

i. z = x == y:

$$\begin{aligned} Out(z) &= In(z) \wedge bool \\ Out(x) &= In(x) \wedge In(y) \\ Out(y) &= In(x) \wedge In(y) \end{aligned}$$

ii. z = x + y:

$$\begin{aligned} Out(z) &= In(z) \wedge int \\ Out(x) &= In(x) \wedge int \\ Out(y) &= In(y) \wedge int \end{aligned}$$

iii. z = x && y:

$$\begin{aligned} Out(z) &= In(z) \wedge bool \\ Out(x) &= In(x) \wedge bool \\ Out(y) &= In(y) \wedge bool \end{aligned}$$

iv. z = int_constant:

$$Out(z) = In(z) \wedge int$$

v. z = bool_constant:

$$Out(z) = In(z) \wedge bool$$

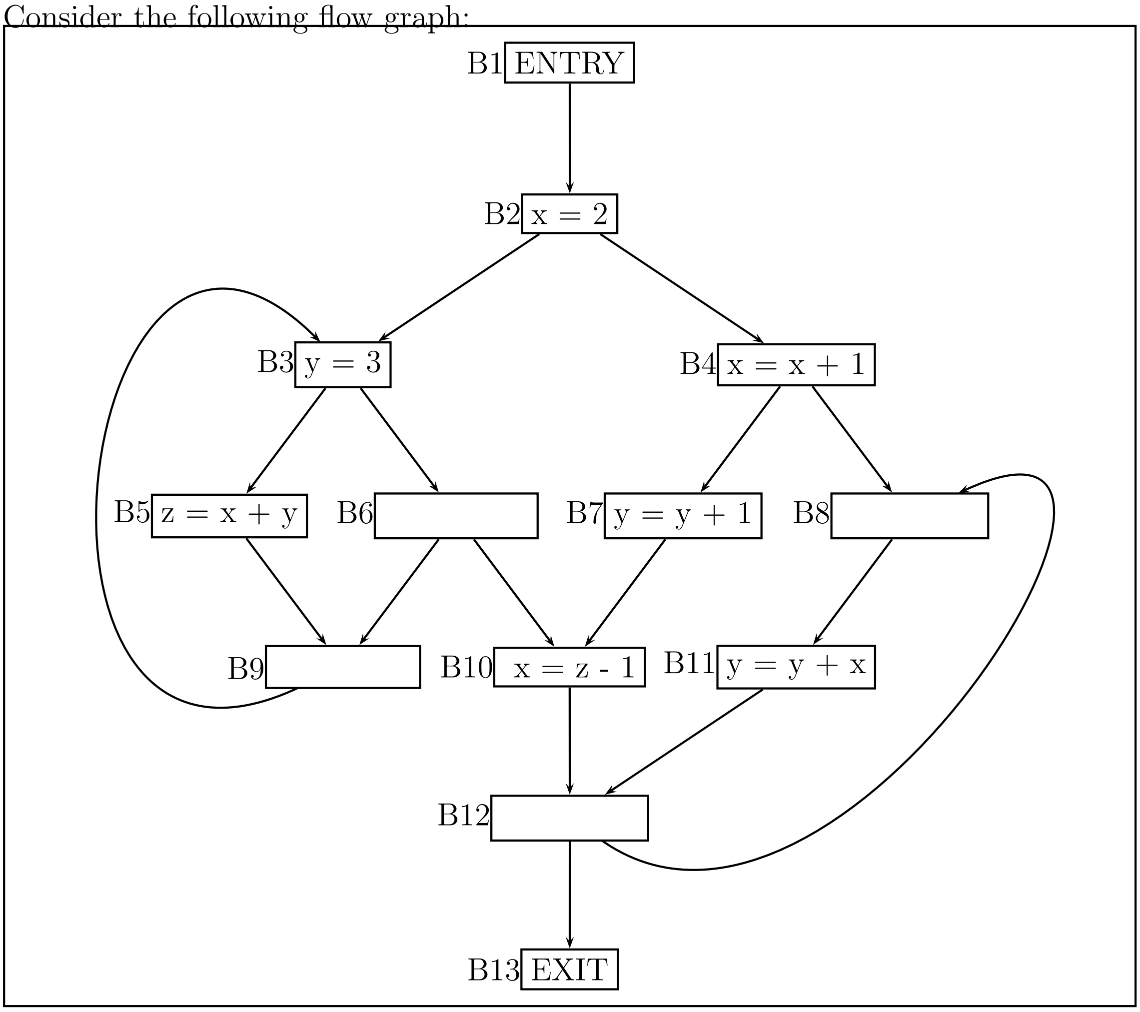
vi. z = x:

$$\begin{aligned} Out(z) &= In(z) \wedge In(x) \\ Out(x) &= In(z) \wedge In(x) \end{aligned}$$

(e) My framework is forward (but yours could be backward, depends on the flow functions!). Out is computed in terms of In.

$$BoundaryInfo = BI, \text{ s.t. } BI(x) = \top \; \forall x$$

4. Consider the following flow graph:



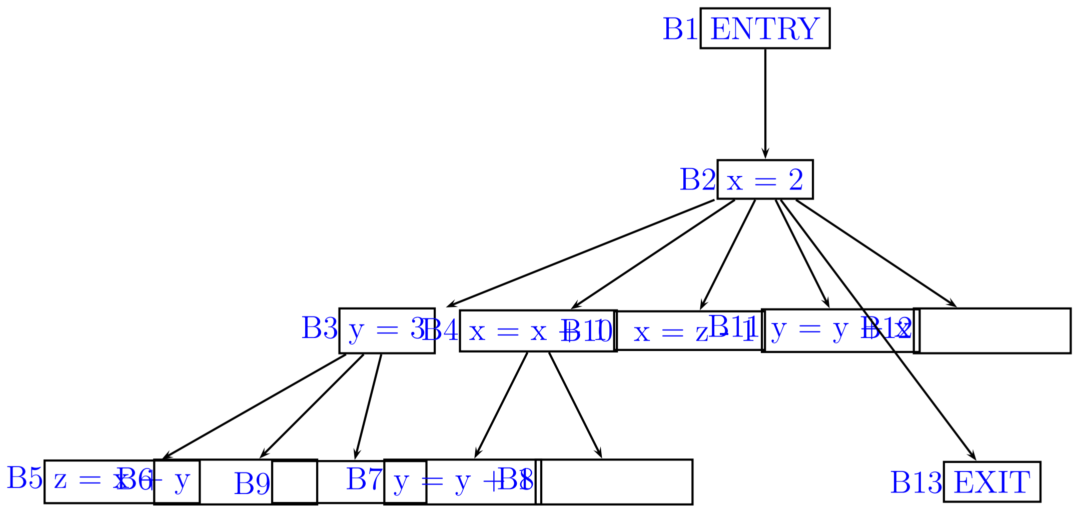
(a) Draw the dominator tree for the graph. [10]

(b) Calculate the dominance frontier for each block. [15]

(c) Calculate the iterated dominance frontiers for the nodes containing the definitions of x, y and z. Assume that ENTRY node (B1) contains implicit definitions of x, y, z as `undef`. [3*5 = 15]

(d) Convert the flow graph to minimal SSA form. [10]

Dominator Tree:

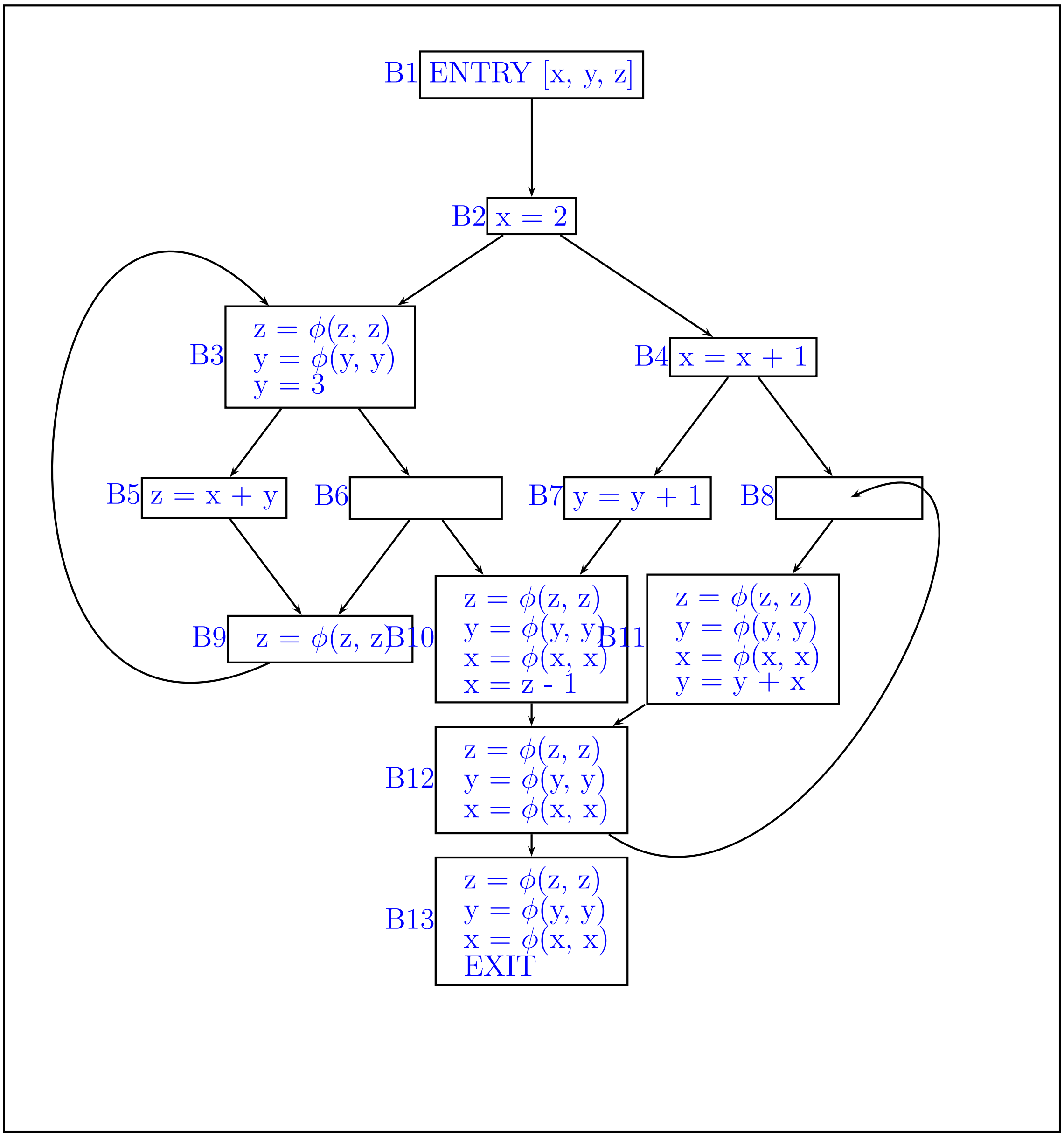


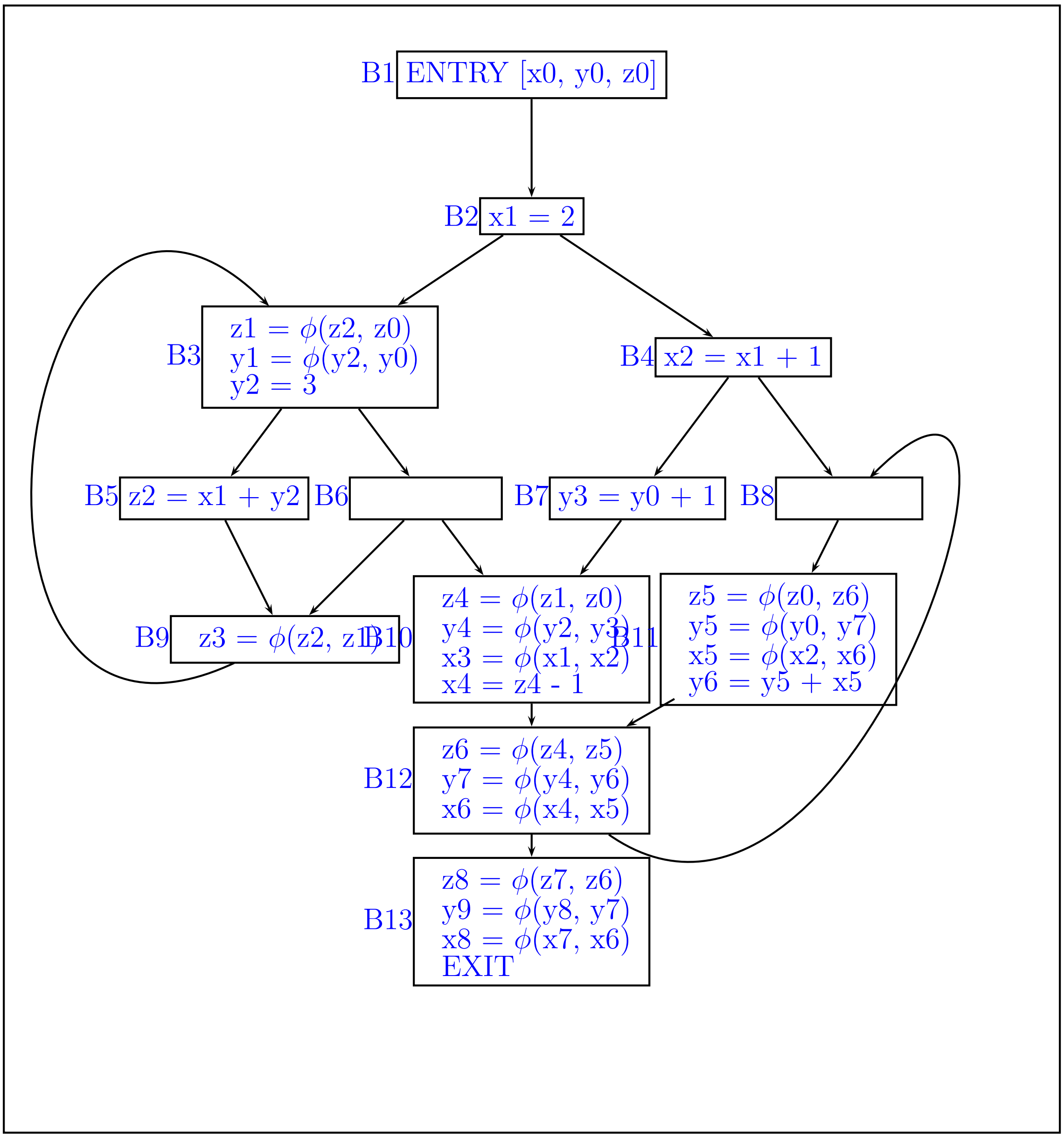
Dominance Frontier:

Node	DF
1	{}
2	{}
3	{3,10,13}
4	{10,11}
5	{3,9}
6	{9,10}
7	{10}
8	{11}
9	{13}
10	{12}
11	{12}
12	{11,13,15}
13	{15}
14	{11,15}
15	{}

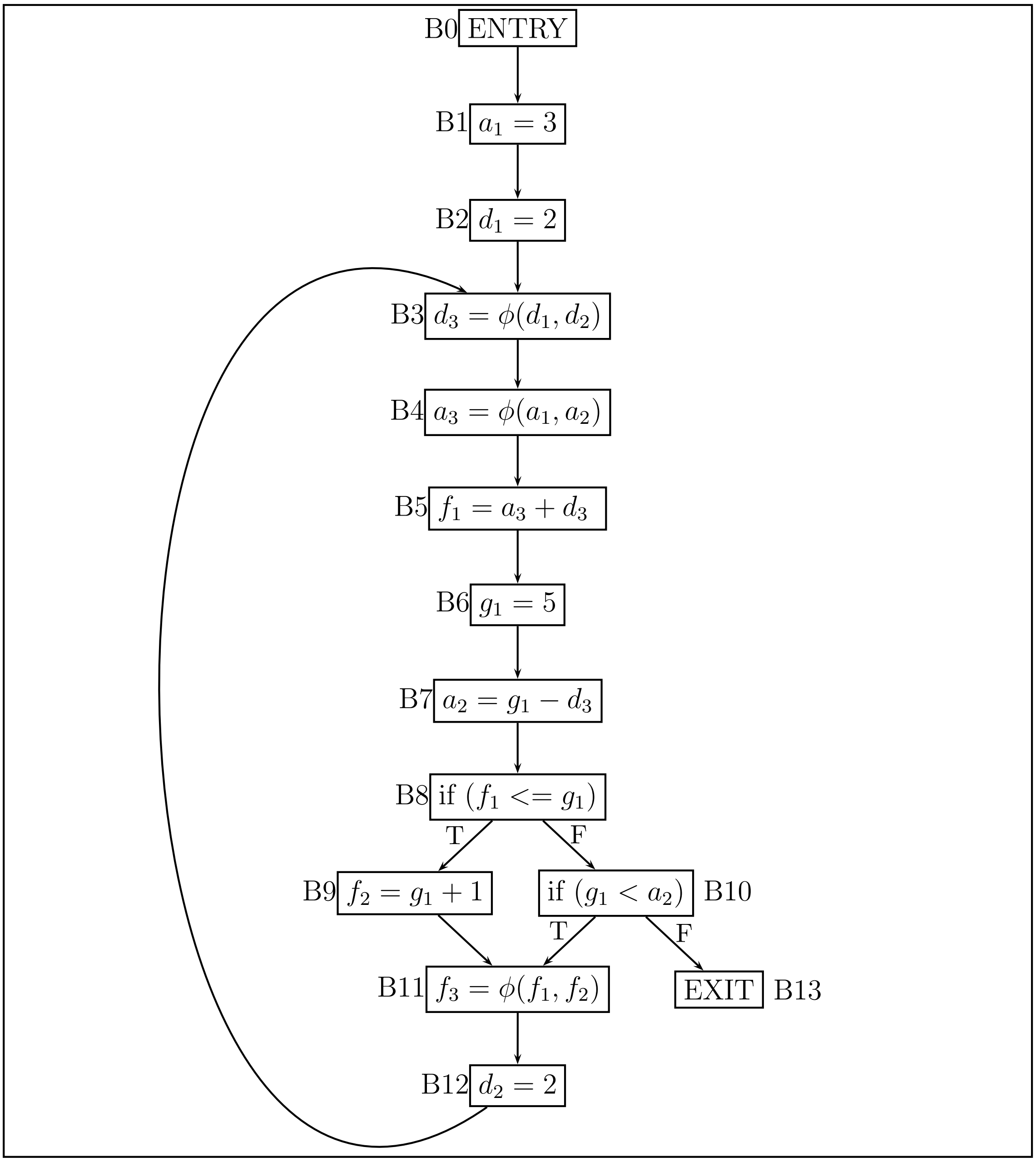
ENTRY node (1) contains implicit definitions of x, y, z.

Var	Defs	Iterated Dom Frontier of Defs
x	1, 2, 4, 10	{ 10, 11, 12, 13, 15 }
y	1, 3, 7, 11	{ 3, 10, 11, 12, 13, 15 }
z	1, 5	{ 3, 9, 10, 11, 12, 13, 15 }


B1 ENTRY [x, y, z]
B2 x = 2
B3 z = φ(z, z)
y = φ(y, y)
y = 3
B4 x = x + 1
B5 z = x + y
B6
B7 y = y + 1
B8
B9 z = φ(z, z)
B10 z = φ(z, z)
y = φ(y, y)
x = φ(x, x)
x = z - 1
B11 z = φ(z, z)
y = φ(y, y)
x = φ(x, x)
y = y + x
B12 z = φ(z, z)
y = φ(y, y)
x = φ(x, x)
B13 z = φ(z, z)
y = φ(y, y)
x = φ(x, x)
EXIT



5. Consider the following flow graph in SSA form, with one instruction per block:



(a) Draw the SSA edges (def-use chains) for the CFG. [5]

(b) Perform Sparse conditional Constant Propagation for the CFG. Clearly show

i. The contents of the worklists (FWL and SWL) at each stage.

ii. The SSA edge or the CFG edge being processed

iii. The values that are updated during processing.

[20]

THE END