NAME: **ROLL NO:**

CS738: Advanced Compiler Optimizations
End Semester Examination, 2011-12

Max Time: 3 Hours Max Marks: 180

NOTE:

- There is no separate answer script. **Write down the answers in the space provided on the question paper.** There is enough space for rough work too, so do not ask for extra sheets.
- 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!
- **ONE** A4 size *cheat-sheet* is allowed. Sharing of cheat-sheet or any other exam material is **not allowed**.
- Last page (#24) contains reproduction of Figures for Ques 4, 5 and 6 to ease cross referencing while writing the answers. You can tear off that page **ONLY after writing your roll number on top of the page.**

Question No.	Max Marks	Marks Obtained
1	10	
2	10	
3	45	
4	35	
5	45	
6	35	
Total	**180**	

I pledge my honour as a gentleman/lady that during the examination I have neither given assistance nor received assistance.

Signature

1. Let $(S, \wedge)$ be a semilattice. Let $f : S \to S$ be a function. Prove that the following two definitions for *monotonicity* of f are equivalent:

$$\forall x, y \in S : f(x \wedge y) \leq f(x) \wedge f(y)$$

and

$$\forall x, y \in S : x \leq y \Rightarrow f(x) \leq f(y)$$

[10]

- $\Rightarrow$: Let $x \leq y$. Then:
  $x \wedge y = x \Rightarrow f(x \wedge y) = f(x)$
  $\Rightarrow f(x) \leq f(x) \wedge f(y)$
  $\Rightarrow f(x) \leq f(y)$ // Recall properties of $\wedge$
- $\Leftarrow$:
  $x \wedge y \leq x \Rightarrow f(x \wedge y) \leq f(x)$
  $x \wedge y \leq y \Rightarrow f(x \wedge y) \leq f(y)$
  Combine the above $\Rightarrow f(x \wedge y) \wedge f(x \wedge y) \leq f(x) \wedge f(y)$
  $\Rightarrow f(x \wedge y) \leq f(x) \wedge f(y)$

2. 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*. [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$\rightarrow$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 $\rightarrow$ X $\rightarrow$ Y $\rightarrow$ 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 $\Rightarrow$ Y $\rightarrow$ Z is a path to Z free of X. But that also contradicts the fact that X dominates Z.

3. Consider a simple programming language with only two primitive types: **int** and **bool**. The language specification does not require a variable to be declared before use. However, the language requires that for a program to be valid, every variable must be assigned a *unique* type during compilation.

   The following table describes the operations permitted by the language, the corresponding type restrictions on the argument variables and the result of the operation.

Operation	**Example**	**Type Restrictions on Args**	**Result Type**
==	x == y	x and y have to be of the same type (int or bool)	bool
+	x + y	x and y both are int	int
&&	x && y	x and y both are bool	bool

   Note that the type of argument on RHS may still be un-inferred when the statement is processed. Further, nested operations are **not allowed** on RHS.

   The language has following constants (each constant has a fixed type):

Const Class	**Constants**	**Type**
int_constant	$-\infty \ldots -2, -1, 0, 1, 2, \ldots \infty$	int
bool_constant	false, true	bool

   The language obviously contains basic constructs like *assignment* statements, *goto* statements, and conditionals (*if-else*). Following typing restrictions apply:

   - Both sides of an assignment must be of the *same* type.
   - The condition of the *if-else* should be a single variable of type bool.

   Here are couple of sample programs, one valid and other invalid.

```
// PROGRAM 1
c = n == 0;
if (c) {
  d = e + 3;
} else {
  d = e - 5;
}
// c: bool
// n, d, e: int
```

```
// PROGRAM 2
c = n==0;
if (c) {
  d  = c + 3;
} else {
  d  = e - 5;
}
// invalid program
// c has to be both bool and int!
// n, d, e: int
```

   Design an **intraprocedural flow-insensitive** data flow analysis framework to infer types of variables for programs written in the above language. In particular,

   (a) Draw the component lattice for the framework. [10]

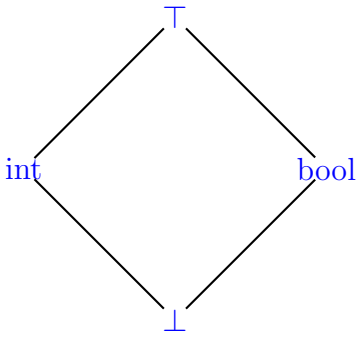
   (b) Describe the meet operator ($\wedge$) in a tabular form. [5]

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

(d) Define the flow functions for each type of statement. [20]

(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:

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

(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. Use call-strings approach to perform *Interprocedural Available Expressions* analysis for the following program. Here c_i denotes the i^{th} call site.

```
main() {
    ENTRY
c1 : call p()
    c * d
c2 : call q()
    EXIT
}
```

```
p() {
    ENTRY
    a * b
    if(...) then
c3 :    call q()
    else
c4 :    call q()
    c = 5
    EXIT
}
```

```
q() {
    ENTRY
    if(...) then
        a = 5
    else
        b = 6
    EXIT
}
```

Show the following steps to arrive at the answer:

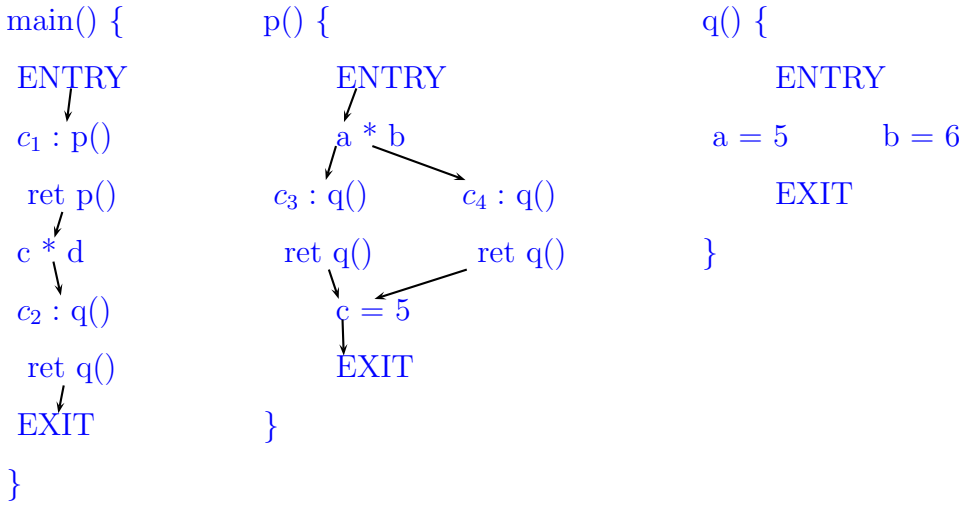
(a) Supergraph distinguishing the intra- and inter- procedural edges.

(b) Local information (gen and kill) for each block

(c) IN and OUT at each program point. The information coming along different call stack should be kept separate.

(d) List the program points where each of the expressions `a * b` and `c * d` is available.

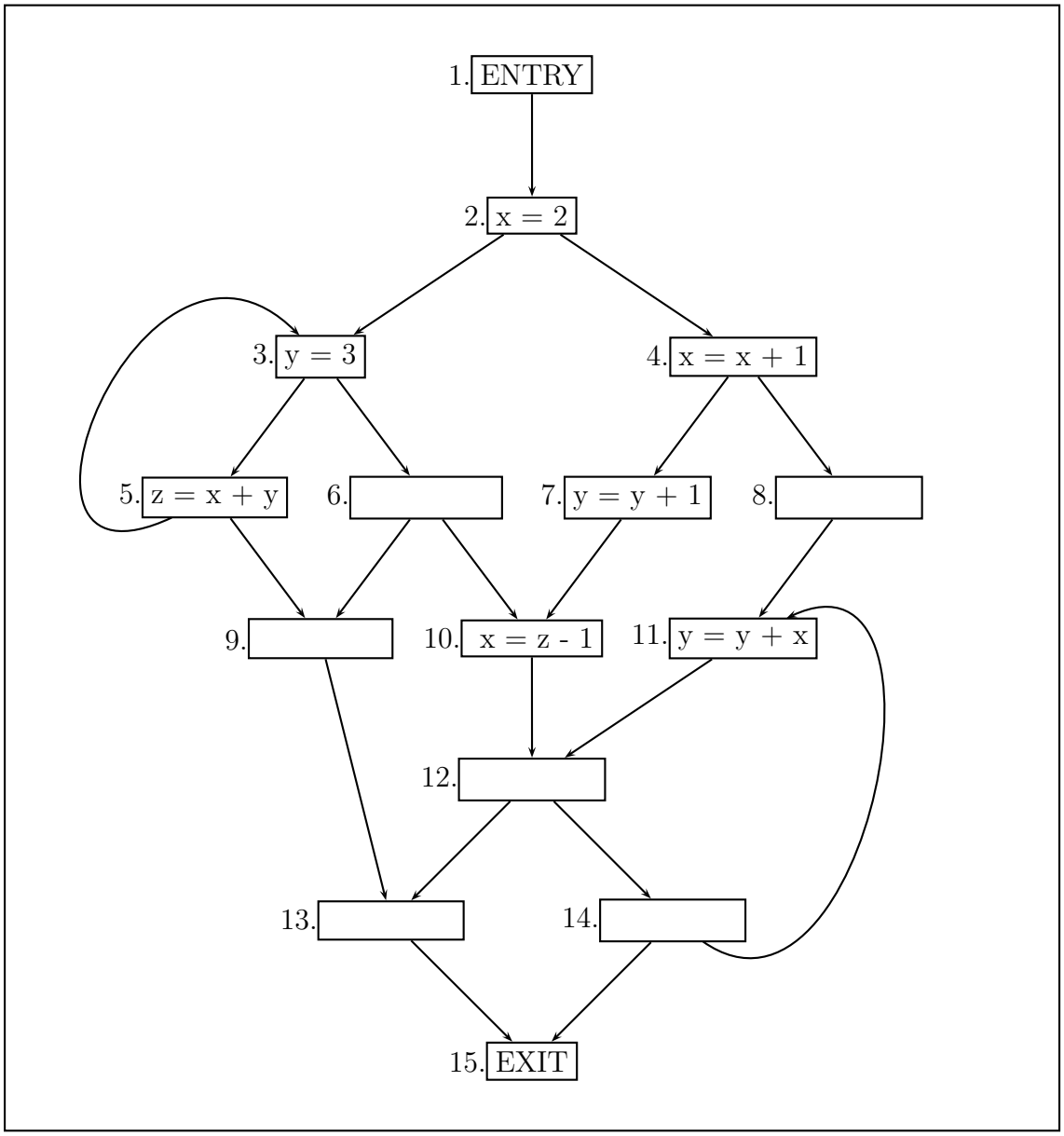
[10+5+15+5]



Pgm	**gen**	**kill**	**IN**	**OUT**
main() {				
ENTRY			$\lambda, 00$	$\lambda, 00$
c_1 : call p()			$\lambda, 00$	$c_1, 00$
c * d	c*d		$\lambda, 00$	$\lambda, 01$
c_2 : call q()			$\lambda, 01$	$c_2, 01$
EXIT			$\lambda, 01$	–
}				
p() {				
ENTRY			$c_1, 00$	$c_1, 00$
a * b	a*b		$c_1, 00$	$c_1, 10$
if(...) then			$c_1, 10$	$c_1, 10$
c_3 : call q()			$c_1, 10$	$c_1c_3, 10$
else			–	–
c_4 : call q()			$c_1, 10$	$c_1c_4, 10$
c = 5		c*d	$c_1, 00$; $c_1, 00$	$c_1, 00$
EXIT			$c_1, 00$	$\lambda, 00$
}				
q() {				
ENTRY			$c_1c_3, 10$; $c_1c_4, 10$; $c_2, 01$	$c_1c_3, 10$; $c_1c_4, 10$; $c_2, 01$
if(...) then			$c_1c_3, 10$; $c_1c_4, 10$; $c_2, 01$	$c_1c_3, 10$; $c_1c_4, 10$; $c_2, 01$
a = 5		a*b	$c_1c_3, 10$; $c_1c_4, 10$; $c_2, 01$	$c_1c_3, 00$; $c_1c_4, 00$; $c_2, 01$
else				
b = 6		a*b	$c_1c_3, 10$; $c_1c_4, 10$; $c_2, 01$	$c_1c_3, 00$; $c_1c_4, 00$; $c_2, 01$
EXIT			$c_1c_3, 00$; $c_1c_4, 00$; $c_2, 01$	$c_1, 00$; $c_1, 00$; $\lambda, 01$
}				

bit1 bit2: bit1 represents availability of a*b, bit2 of c*d you can also initialize boundary with $\perp$ instead of 0. Meet at each point, ignoring call string part will give the availability.

5. Consider the following flow graph:

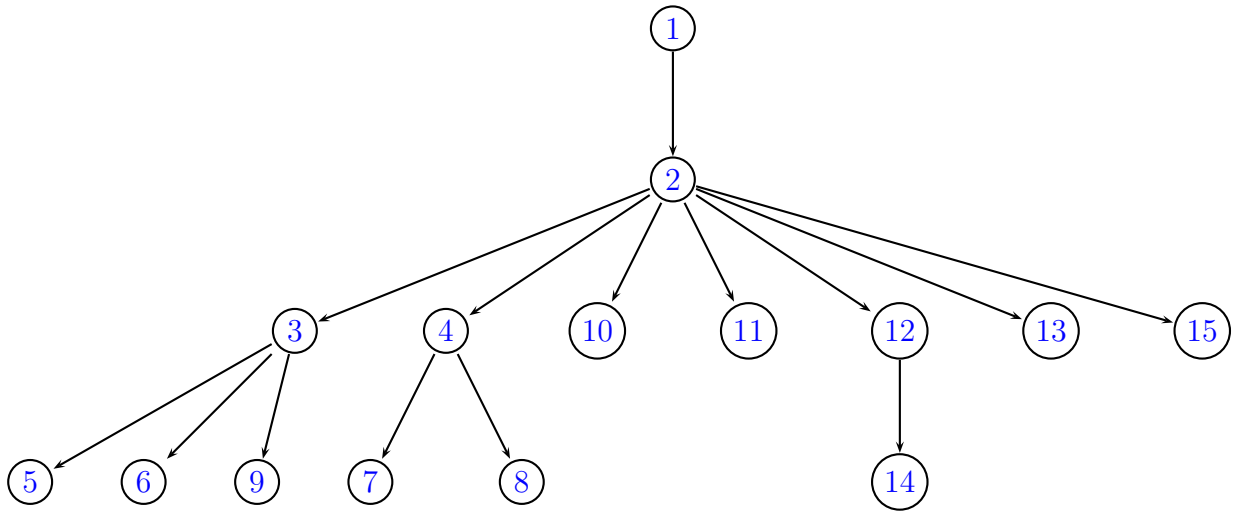


(a) Draw the dominator tree for the graph. [5]

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

(c) Convert the flow graph to minimal SSA form. Show the important steps in conversion. [25]

Dominator Tree:

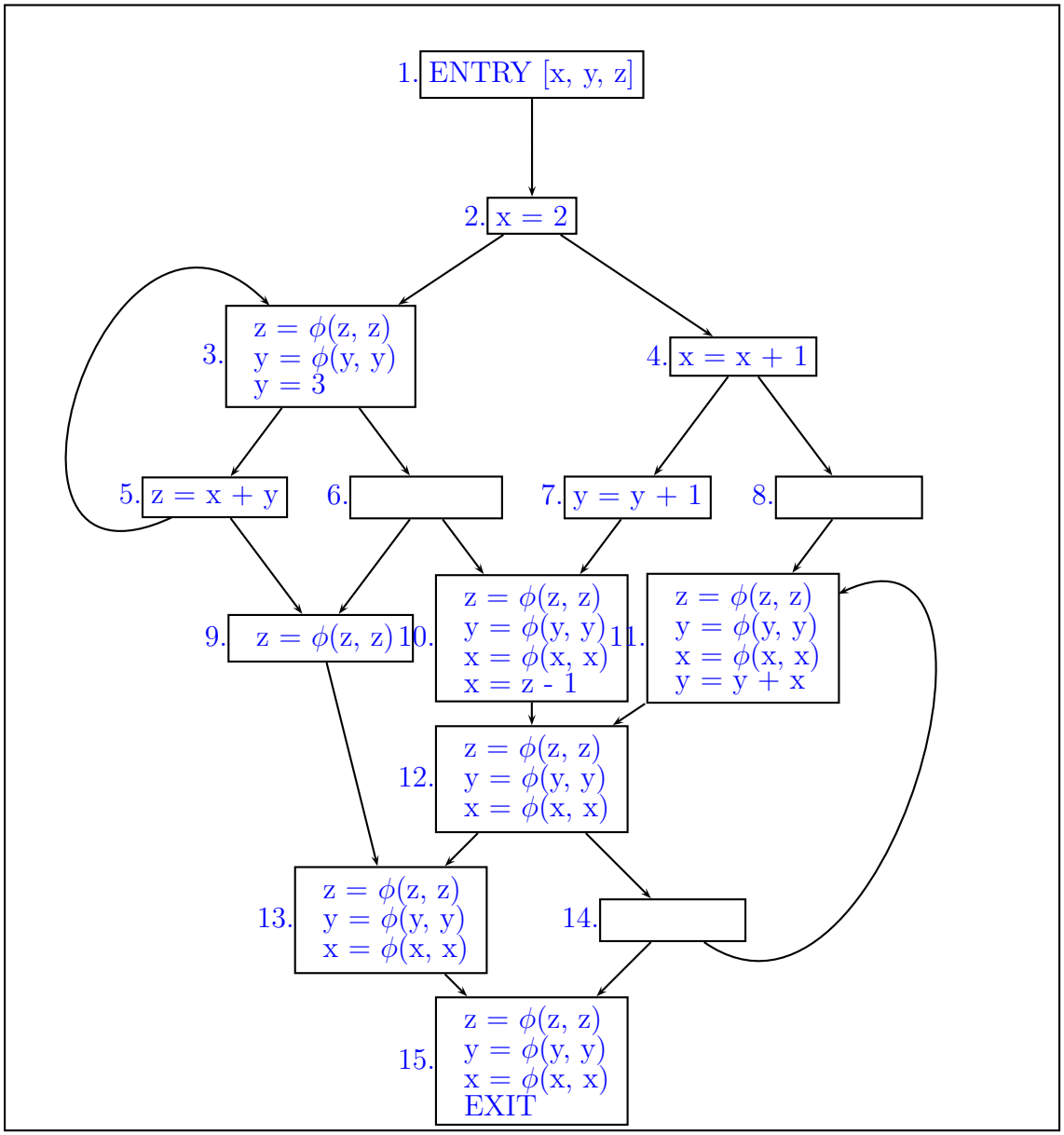


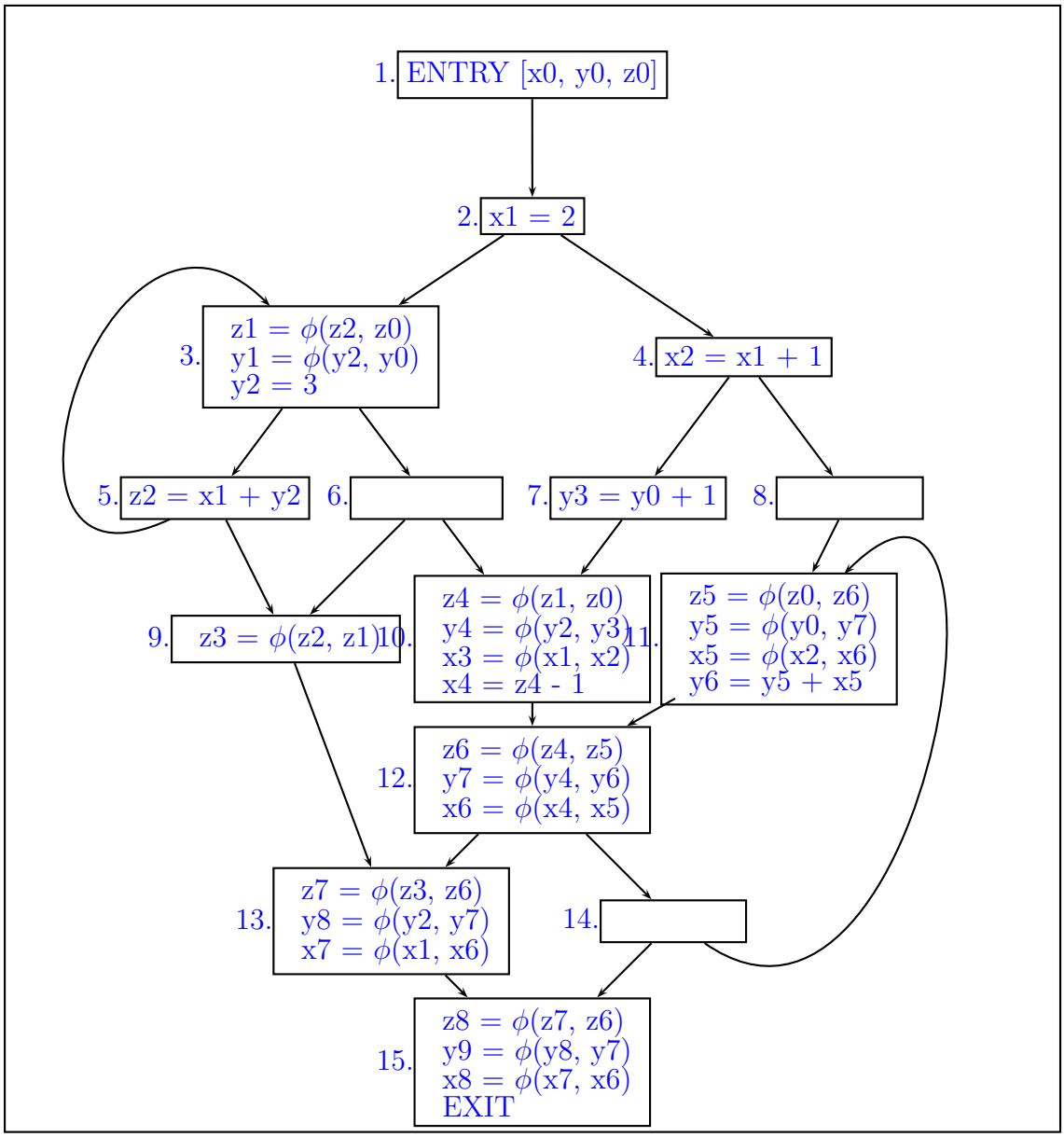
Dominance Frontier:

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 }



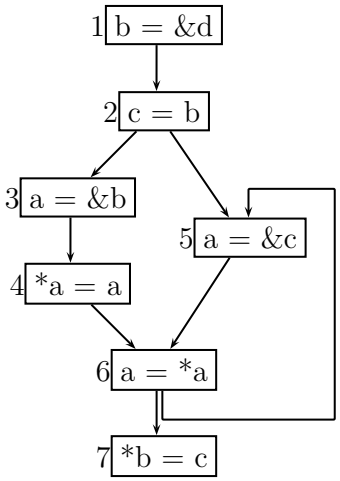

1. ENTRY [x0, y0, z0]
2. x1 = 2
3. z1 = φ(z2, z0)
y1 = φ(y2, y0)
y2 = 3
4. x2 = x1 + 1
5. z2 = x1 + y2
6.
7. y3 = y0 + 1
8.
9. z3 = φ(z2, z1)
10. z4 = φ(z1, z0)
y4 = φ(y2, y3)
x3 = φ(x1, x2)
x4 = z4 - 1
11. z5 = φ(z0, z6)
y5 = φ(y0, y7)
x5 = φ(x2, x6)
y6 = y5 + x5
12. z6 = φ(z4, z5)
y7 = φ(y4, y6)
x6 = φ(x4, x5)
13. z7 = φ(z3, z6)
y8 = φ(y2, y7)
x7 = φ(x1, x6)
14.
15. z8 = φ(z7, z6)
y9 = φ(y8, y7)
x8 = φ(x7, x6)
EXIT

6. Perform **Flow Sensitive May and Must Points-to** analysis for the program.

   (a) Give Local Data Flow information for each block in terms of May and Must sets. [Do not write the generic equation, but instantiate it for the current statement.]

   (b) Show result of each iteration of the analysis (Maximum **3** iterations only).

   The final answer should be given in the tables provided below. [10 + 25]



#	May_{GEN}	May_{KILL}	$Must_{GEN}$	$Must_{KILL}$
1	$b \rightarrow d$	$b \rightarrow any$	$b \rightarrow d$	$b \rightarrow any$
2	$c \rightarrow p$ $\|b \rightarrow p \in MayIN$	$c \rightarrow any$	$c \rightarrow p$ $\|b \rightarrow p \in MustIN$	$c \rightarrow any$
3	$a \rightarrow b$	$a \rightarrow any$	$a \rightarrow b$	$a \rightarrow any$
4	$p \rightarrow p'$ $\|a \rightarrow p \in MayIN$ $\|a \rightarrow p' \in MayIN$	$p \rightarrow any$ $\|a \rightarrow p \in MustIn$	$p \rightarrow p'$ $\|a \rightarrow p \in MustIN$ $\|a \rightarrow p' \in MustIN$	$p \rightarrow any$ $\|a \rightarrow p \in MayIn$
5	$a \rightarrow c$	$a \rightarrow any$	$a \rightarrow c$	$a \rightarrow any$
6	$a \rightarrow p$ $\|a \rightarrow p' \in MayIN$ $\|p' \rightarrow p \in MayIN$	$a \rightarrow any$	$a \rightarrow p$ $\|a \rightarrow p' \in MustIN$ $\|p' \rightarrow p \in MustIN$	$a \rightarrow any$
7	$p \rightarrow p'$ $\|b \rightarrow p \in MayIN$ $\|c \rightarrow p' \in MayIN$	$p \rightarrow any$ $\|b \rightarrow p \in MustIn$	$p \rightarrow p'$ $\|b \rightarrow p \in MustIN$ $\|c \rightarrow p' \in MustIN$	$p \rightarrow any$ $\|b \rightarrow p \in MayIn$

Local Data Flow Information

#	May_{IN}	May_{OUT}	$Must_{IN}$	$Must_{OUT}$
1	ϕ	$b \to d$	ϕ	$b \to d$
2	$b \to d$	$b \to d, c \to d$	$b \to d$	$b \to d, c \to d$
3	$b \to d, c \to d$	$a \to b, b \to d, c \to d$	$b \to d, c \to d$	$a \to b, b \to d, c \to d$
4	$a \to b, b \to d, c \to d$	$a \to b, b \to b, c \to d$	$a \to b, b \to d, c \to d$	$a \to b, b \to b, c \to d$
5	$b \to d, c \to d$	$a \to c, b \to d, c \to d$	$b \to d, c \to d$	$a \to c, b \to d, c \to d$
6	$a \to bc, b \to bd, c \to d$	$a \to bd, b \to bd, c \to d$	$c \to d$	$c \to d$
7	$a \to bd, b \to bd, c \to d$	$a \to bd, b \to bd, c \to d$ $d \to d$	$c \to d$	$c \to d$

Result of Analysis for Iteration No. 1

#	May_{IN}	May_{OUT}	$Must_{IN}$	$Must_{OUT}$
1	ϕ	$b \to d$	ϕ	$b \to d$
2	$b \to d$	$b \to d, c \to d$	$b \to d$	$b \to d, c \to d$
3	$b \to d, c \to d$	$a \to b, b \to d, c \to d$	$b \to d, c \to d$	$a \to b, b \to d, c \to d$
4	$a \to b, b \to d, c \to d$	$a \to b, b \to b, c \to d$	$a \to b, b \to d, c \to d$	$a \to b, b \to b, c \to d$
5	$\mathbf{a \to bd, b \to bd, c \to d}$	$\mathbf{a \to c, b \to bd, c \to d}$	$\mathbf{c \to d}$	$\mathbf{a \to c, c \to d}$
6	$a \to bc, b \to bd, c \to d$	$a \to bd, b \to bd, c \to d$	$c \to d$	$c \to d$
7	$a \to bd, b \to bd, c \to d$	$a \to bd, b \to bd, c \to d$ $d \to d$	$c \to d$	$c \to d$

Result of Analysis for Iteration No. 2

#	May_{IN}	May_{OUT}	$Must_{IN}$	$Must_{OUT}$
1	ϕ	$b \to d$	ϕ	$b \to d$
2	$b \to d$	$b \to d, c \to d$	$b \to d$	$b \to d, c \to d$
3	$b \to d, c \to d$	$a \to b, b \to d, c \to d$	$b \to d, c \to d$	$a \to b, b \to d, c \to d$
4	$a \to b, b \to d, c \to d$	$a \to b, b \to b, c \to d$	$a \to b, b \to d, c \to d$	$a \to b, b \to b, c \to d$
5	$a \to bd, b \to bd, c \to d$	$a \to c, b \to bd, c \to d$	$c \to d$	$a \to c, c \to d$
6	$a \to bc, b \to bd, c \to d$	$a \to bd, b \to bd, c \to d$	$c \to d$	$c \to d$
7	$a \to bd, b \to bd, c \to d$	$a \to bd, b \to bd, c \to d$ $d \to d$	$c \to d$	$c \to d$

Result of Analysis for Iteration No. 3 = No. 2