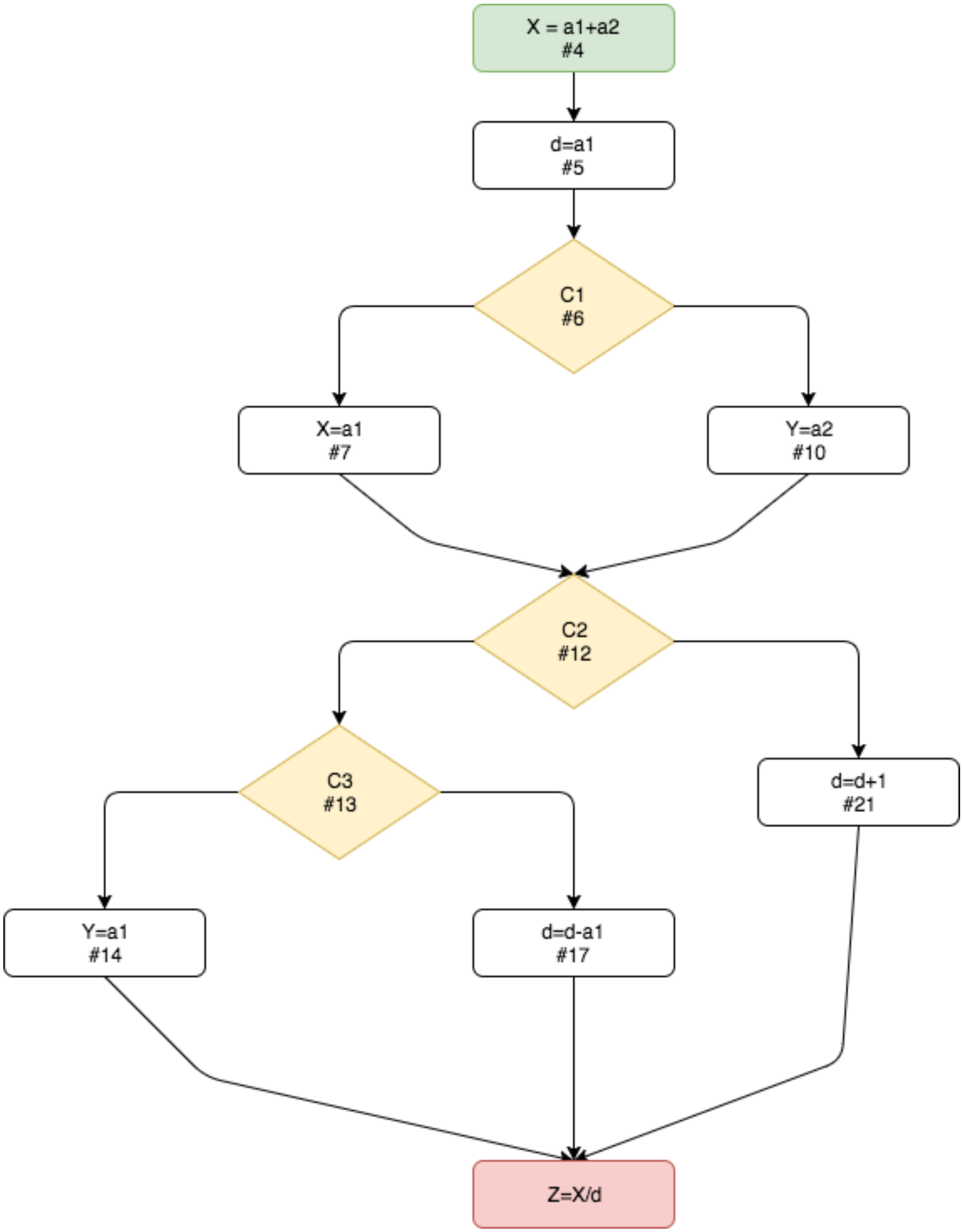
Homework 1 (20 points), SE421, 1/18/2018, due: Friday, 1/19/2018
Schreck, Mason

Problem 1
Part A (3 points): Draw the CFG (If the CFG is wrong, your subsequent answers are likely to be wrong. Make sure that your CFG is correct. You can actually find a similar CFG in the lecture notes.



Homework 1 (20 points), SE421, 1/18/2018, due: Friday, 1/19/2018
Schreck, Mason

Note: The starting statement we were given has been colored green, all conditionals yellow and the terminating statement in red. And the line number denoted as #N, e.g. the ending node is #23 (it got cutoff).

Part B (2 points): Count the number of paths in the CFG.
6 paths within the CFG – assuming each path is a unique way to get from the green (start node) to the red (end node) of the CFG above.

Part C (3 points): Give the execution trace for each path using the line numbers (e.g. 4,5,6,7,12,13,14,23).

Path#:
1. 4,5,6,7,12,13,14,23
2. 4,5,6,7,12,13,17,23
3. 4,5,6,7,12,21,23
4. 4,5,6,10,12,13,14,23
5. 4,5,6,10,12,13,17,23
6. 4,5,6,10,12,21,23

Part D (3 points): Give the corresponding Boolean values of each execution trace (e.g. For the above trace, it is C1 == TRUE, C2 == TRUE, C3 == TRUE). To answer this question, number the traces and use that numbering to give the corresponding Boolean values for C1, C2, and C3.

Path#:
1. C1==TRUE, C2==TRUE, C3==TRUE
2. C1==TRUE, C2==TRUE, C3==FALSE
3. C1==TRUE, C2==FALSE
4. C1==FALSE, C2==TRUE, C3==TRUE
5. C1==FALSE, C2==TRUE, C3==FALSE
6. C1==FALSE, C2==FALSE

Part E (2 points): Give the line numbers for the program statements that are relevant for analyzing the division-by-zero (DBZ) vulnerability on line 23.
This will be any line that manipulates the value of 'd'. I will not be considering line 5 in this problem since that is assumed initialization for this problem. Therefore the only lines that influence the value of 'd' are lines 17 and 21.

Part F (3 points): Give the relevant traces for each path (e.g. 5, 12, 13, 23 is the relevant trace for the execution trace given above).

Path#:
1. 5,12,13,23
2. 5,12,13,17,23
3. 5,12,21,23
4. 5,12,13,23
5. 5,12,13,17,23
6. 5,12,21,23

> **Note:** Following the unique paths and the unique Boolean values, we get repeated relevant traces.

Part G (2 points): Count the number of relevant traces.
Following the note I left in part F of this problem, there are in fact only three *unique* relevant traces.

Part H (2 points): Give the relevant traces that actually exhibit the DBZ vulnerability.
For part F I chose paths where the nodes could only directly influence the value of d along the trace. From the email received "The relevant trace is a subsequence of the trace that consists of only those statements along the path that is relevant to division-by-zero (DBZ)." Now to know along which trace exactly the DBZ issue will occur we would need to know the starting values of a1, a2 so we can trace along which path d will in fact be zero. So to answer the next question I will assume that only one of the six paths lead to a DBZ error and the relevant trace for that is:

> 5,12,13,17,23

I made this assumption since this is the only relevant path that leads to a decrement of d.

Part I (2 points): How many times the program must be run to actually observe the DBZ vulnerability with probability 0.9 or above? Assume that each path in the CFG is equally likely. (Hint: if *p* is the probability to *not* observe the vulnerability in one program run then the probability of observing the vulnerability in *n* program runs is $1\text{-}p^n$).
Based upon the assumption in part H of the question I will say that 5/6 paths will lead to a flawless run, and only 1/6 paths lead to a DBZ error. This leads to setting up the equation as:

$$0.9 = 1-(5/6)^n$$

[1] $0.9 = 1 - (0.83333)^n$

[2] $0.9 - 1 = - (0.83333)^n$

[3] $- 0.1 = - 0.83333^n$

[4] $0.1 = 0.83333^n$

[5] $\ln(0.1) = n * \ln(0.83333)$

[6] $n = \ln(0.1) / \ln(0.83333)$

[7] n is approximately 12.62897

^{8} Rounding this up, I will say that it will take around 13 program executions to observe

Problem 2:
Assuming that the return statement isn't supposed to actually be a programmatic return (I don't see "return" anywhere in the code snippet) then it is correct.