David Dae Ho Kim
20415096
CS 447 - 001
dhdkim@uwaterloo.ca

Question 1
c++ output: -1
java output: -1
perl output: 1
python output: 1

Java and c++ takes the sign of the dividend when performing modulo operations.
To cope with such discrepancies, a standard should be placed for developers to take these into account. They should take precautions so that these errors may be avoided. Another strategy is to modify the compilers such that a standard is in place for compilers as well. When a compiler observes that such error may occur, it should automatically compile such that these errors can be avoided.

Question 2
a)
test: x = NULL
null value for x will result in a NULLPOINTEREXCEPTION and will not execute the fault.

b)
test: x = []
This empty case will execute the fault, but will not result in an error state

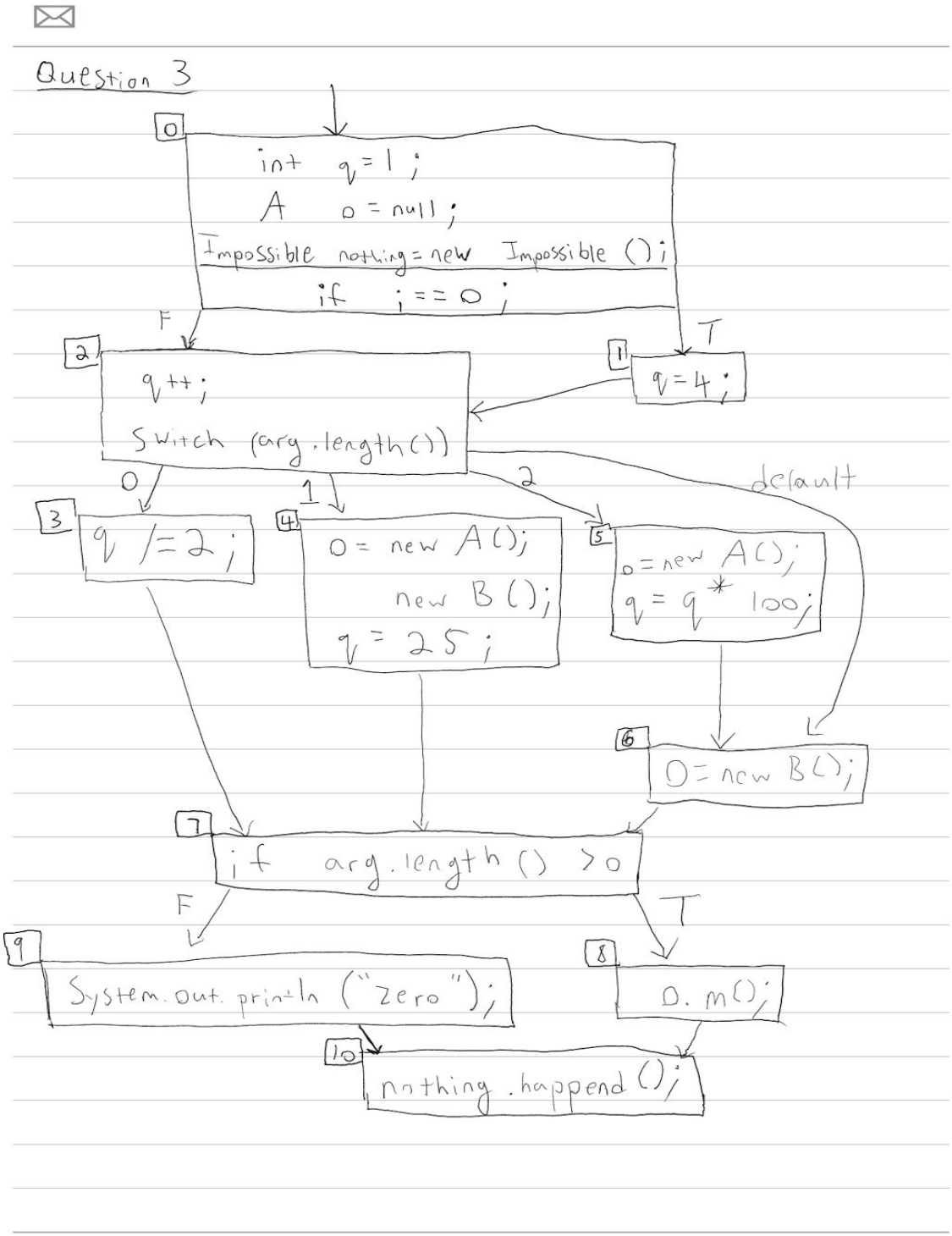
c)
It is impossible to show this

d)
Error State
x = [-10, -9, 0, 99, 100]
i = 1
count = 0
PC = i++

expected: 2
actual: 1

Question 3

a)


Question 3
0
int q = 1;
A o = null;
Impossible nothing = new Impossible ();
if i == 0;
F
T
2
q++;
switch (arg.length())
1
q = 4;
0
1
2
default
3
q /= 2;
4
o = new A();
new B();
q = 25;
5
o = new A();
q = q * 100;
6
O = new B();
7
if arg.length() > 0
F
T
9
System.out.println("zero");
8
O.m();
10
nothing.happend();

b)
Node Coverage:

TR = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Test Paths = [0, 1, 2, 3, 7, 9, 10]
[0, 2, 4, 7, 8, 10]
[0, 2, 5, 6, 7, 8, 10]

Edge Coverage:

TR = {[0,1],[0,2],[1,2],
[2,3],[2,4],[2,5],
[2,6],[3,7],[4,7],
[5,6],[6,7],[7,9],
[7,8],[9,10],[8,10]}

Edge Pair Coverage:

TR = {[0,1,2],[0,2,3],[1,2,3],
[1,2,4],[1,2,5],[1,2,6],
[2,3,7],[2,4,7],[2,5,6],
[2,6,7],[3,7,9],[4,7,8],
[5,6,7], [6,7,8],[7,9,10],
[7,8,10]}
[3,7,8],<-- infeasible because node 3 states arg.length() == 0, but node 8 states arg.length() > 0
[4,7,9],<-- infeasible because node 4 states arg.length() == 1, but node 9 states arg.length() == 0
[6,7,9],<-- infeasible because node 6 states arg.length() >= 2, but node 9 states arg.length() == 0

Prime Path Coverage:

TR = {[0,1,2,5,6,7,8,10],
[0,1,2,4,7,8,10],
[0,1,2,3,7,8,10],
[0,2,3,7,9,10],
[0,2,4,7,8,10],
[0,2,5,6,7,8,10],
[0,1,2,6,7,8,10],
[0,2,6,7,8,10]}

[0,1,2,5,6,7,9,10],<-- infeasible because to get to node 5, arg.length() == 2, but in node 9 we're saying arg.length() == 0
[0,1,2,4,7,9,10],<-- infeasible because to get to node 4, arg.length() == 1, but in node 9 we're saying arg.length() == 0
[0,1,2,3,7,8,10],<-- infeasible because to get to node 3, arg.length() == 0, but in node 8 we're saying arg.length() > 0
[0,2,3,7,8,10], <-- infeasible because node 3 states arg.length() == 0, but node 8 states arg.length() > 0
[0,2,4,7,9,10], <-- infeasible because node 4 states arg.length() == 1, but node 9 states arg.length() == 0
[0,2,5,6,7,9,10],<--infeasible because node 5 states arg.length() == 2, but node 9 states arg.length() == 0
[0,1,2,6,7,9,10],<--infeasible because node 6 states arg.length() > 2, but node 9 states arg.length() == 0
[0,2,6,7,9,10], <--infeasible because node 6 states arg.length() > 2, but node 9 states arg.length() == 0