Q1:

1. Truth Table

Rows	a	b	c	¬b	¬b ∨ c	p=a ∧ (¬b ∨ c)
1	T	T	T	F	T	T
2	T	T	F	F	F	F
3	T	F	T	T	T	T
4	T	F	F	T	T	T
5	F	T	T	F	T	F
6	F	T	F	F	F	F
7	F	F	T	T	T	F
8	F	F	F	T	T	F

Please enter your logic expression in the text box below, using the following logic operators. The variables can be any string. Parentheses can be used in the logic expression. (Please be aware that this application gets slow for expressions that have more than 5 or 6 variables.)

Not : **!** And : **&** Or : |

Implication : > Exclusive Or : ^ Equivalence : =

P = a & (!b | c)

Test Requirements: Truth Table

Tests: GACC CACC RACC GICC RICC

Others: New Expression Graph Coverage Data Flow Coverage Minimal-MUMCUT Coverage

Share Expression: Share

Truth Table:

Row#	a	b	c	P	Pa	Pb	Pc
1	T	T	T	T	T		T
2	T	T				T	T
3	T		T	T	T		
4	T			T	T	T	
5		T	T		T		
6		T					
7			T		T		
8					T		

2.
Clause a:
Clause a determines p if **¬b ∨ c is fixed** (always T or F).
- If fix ¬b ∨ c = T then: p = a = T
- If fix ¬b ∨ c = F then: p = F

Clause b:
Clause b determines p if **a = T and c is fixed**.
- If fix a = T, c = T then: p = T
- If fix a = T, c = F then: p = ¬b = F

Clause c:
Clause c determines p if **a = T and b is fixed**.
- Fix a = T, b = T: p = c.
- Fix a = T, b = F: p = T.

3.
General Active Clause Coverage (GACC): Each major clause must be both true and false while the minor clauses are fixed to some values.
Example test pairs:
- a=1,b=1,c=1 → p=1
- a=0,b=1,c=1 → p=0
- (Similar pairs for b and c)

Correlated Active Clause Coverage (CACC): The major clause must independently affect p.
Example:
- a=1,b=1,c=1 (p = 1)
- a=0,b=1,c=1 (p = 0)

Restricted Active Clause Coverage (RACC): Fix non-major clauses to ensure independence.

General Inactive Clause Coverage (GICC): Each clause should not affect p.

Restricted Inactive Clause Coverage (RICC): Similar to GICC but with more constraints.

Clause a
- Active (Satisfies GACC, CACC and RACC)
  - (T, F, F) vs. (F, F, F) when (b, c) = (F, F)
  - (T, F, T) vs. (F, F, T) when (b, c) = (F, T)
  - (T, T, T) vs. (F, T, T) when (b, c) = (T, F)
- Inactive (Satisfies GICC and RICC)
  - (T, T, F) vs. (F, T, F) when (b, c) = (T, F)

Clause b

- Active (Satisfies GACC, CACC and RACC)
  - (T, T, F) vs. (T, F, F) when (a, c) = (T, F)
- Inactive (Satisfies GICC and RICC)
  - (T, T, T) vs. (T, F, T) when (a, b) = (T, inactive)

Clause c

- Active (Satisfies GACC, CACC and RACC)
  - (T, T, T) vs. (T, T, F) when (a, b) = (T, T)
- Inactive (Satisfies GICC and RICC)
  - (T, F, T) vs. (T, F, F) when (a, c) = (T, inactive)

The following result for GACC is based on the truth table on the right:

Major Clause	Set of possible tests
a	(1,5), (1,7), (1,8), (3,5), (3,7), (3,8), (4,5), (4,7), (4,8)
b	(2,4)
c	(1,2)

Truth Table:

Row#	a	b	c	P	Pa	Pb	Pc
1	T	T	T	T	T		T
2	T	T				T	T
3	T		T	T	T		
4	T			T	T	T	
5		T	T		T		
6		T					
7			T		T		
8					T		

The following result for CACC is based on the truth table on the right:

Major Clause	Set of possible tests
a	(1,5), (1,7), (1,8), (3,5), (3,7), (3,8), (4,5), (4,7), (4,8)
b	(2,4)
c	(1,2)

Truth Table:

Row#	a	b	c	P	Pa	Pb	Pc
1	T	T	T	T	T		T
2	T	T				T	T
3	T		T	T	T		
4	T			T	T	T	
5		T	T		T		
6		T					
7			T		T		
8					T		

The following result for RACC is based on the truth table on the right:

Major Clause	Set of possible tests
a	(1,5), (3,7), (4,8)
b	(2,4)
c	(1,2)

Truth Table:

Row#	a	b	c	P	Pa	Pb	Pc
1	T	T	T	T	T		T
2	T	T				T	T
3	T		T	T	T		
4	T			T	T	T	
5		T	T		T		
6		T					
7			T		T		
8					T		

The following result for GICC is based on the truth table on the right:

Major Clause	Set of possible tests	
a	No feasible pairs for P = T	P = F: (2,6)
b	P = T: (1,3)	P = F: (5,7), (5,8), (6,7), (6,8)
c	P = T: (3,4)	P = F: (5,6), (5,8), (7,6), (7,8)

Truth Table:

Row#	a	b	c	P	Pa	Pb	Pc
1	T	T	T	T	T		T
2	T	T				T	T
3	T		T	T	T		
4	T			T	T	T	
5		T	T		T		
6		T					
7			T		T		
8					T		

The following result for RICC is based on the truth table on the right:

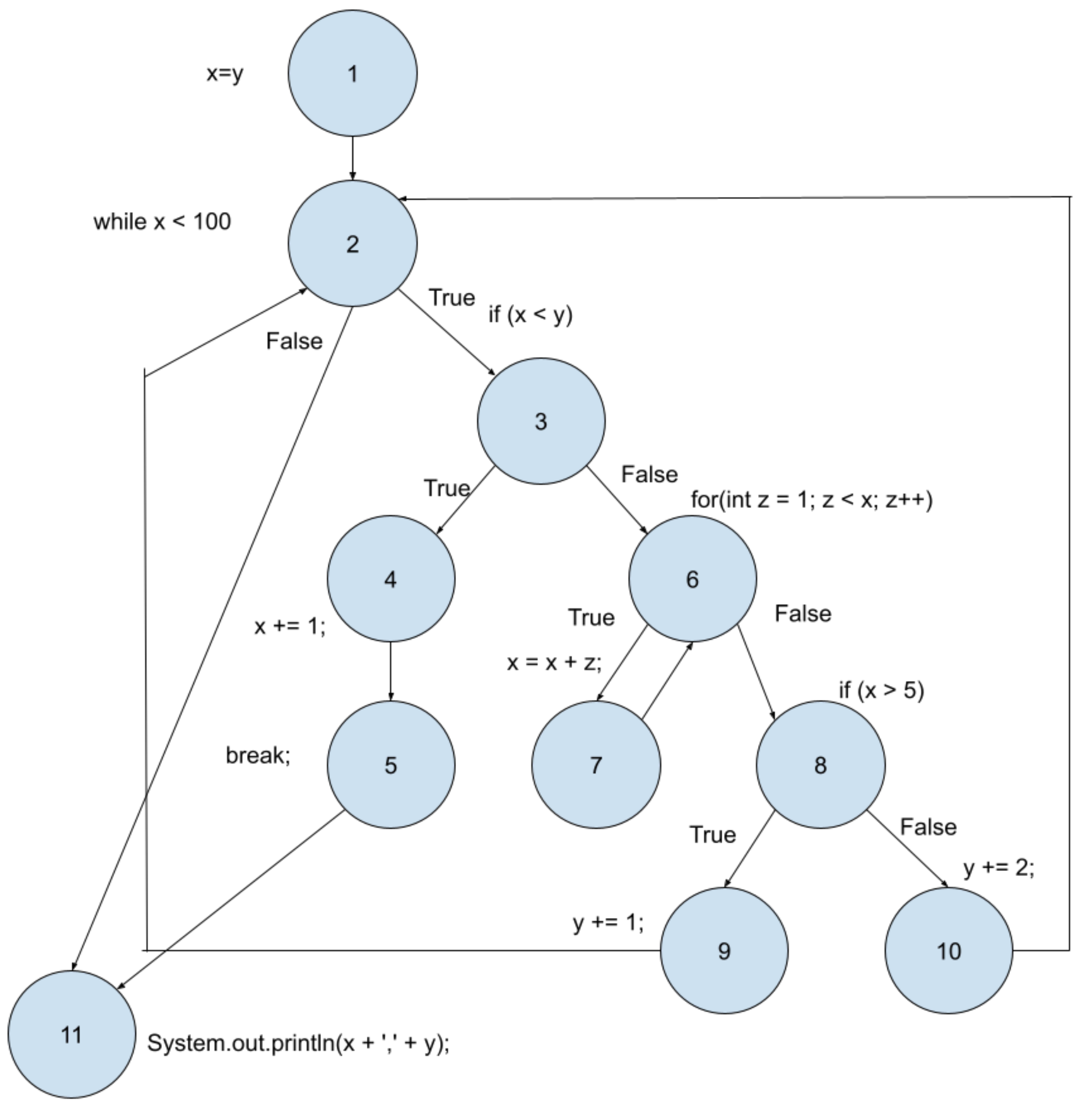
Major Clause	Set of possible tests	
a	No feasible pairs for P = T	P = F: (2,6)
b	P = T: (1,3)	P = F: (5,7), (6,8)
c	P = T: (3,4)	P = F: (5,6), (7,8)

Truth Table:

Row#	a	b	c	P	Pa	Pb	Pc
1	T	T	T	T	T		T
2	T	T				T	T
3	T		T	T	T		
4	T			T	T	T	
5		T	T		T		
6		T					
7			T		T		
8					T		

Q2:

1. CFG



2.
Node 1: def(1) = {x}, use(1) = {y}
Node 2: def(2) = {}, use(2) = {x}
Node 3: def(3) = {}, use(3) = {x, y}
Node 4: def(4) = {x}, use(4) = {x}
Node 5: def(5) = {}, use(5) = {}
Node 6: def(6) = {z}, use(6) = {x}
Node 7: def(7) = {x}, use(7) = {x, z}
Node 8: def(8) = {}, use(8) = {x}
Node 9: def(9) = {y}, use(9) = {y}
Node 10: def(10) = {y}, use(10) = {y}

Node 11: def(11) = {}, use(11) = {x, y}

3.
For variable x:

- (1, 2): def at node 1, use at node 2
- (1, 3): def at node 1, use at node 3
- (4, 11): def at node 4, use at node 11
- (7, 8): def at node 7, use at node 8
- (7, 11): def at node 7, use at node 11

For variable y:

- (9, 11): def at node 9, use at node 11
- (10, 11): def at node 10, use at node 11

For variable z:

- (6, 7): def at node 6, use at node 7

4.

1. Path including nodes 1 → 2 → 3 → 4 → 5 → 11 (break statement skips the rest of the loop).
2. Path including only one iteration of the for loop (6→7→6) without checking node 8.
3. Path 3 (having $x < 100$ true but $x > 5$ false after updates that make $x \geq 6$) is infeasible because once x starts at 6 (or is incremented to 6 or more), the check $x > 5$ will be true.
4. A path that attempts both the if ($x < y$) = true branch and the for loop in the same iteration (i.e., Node 3 → Node 4 → Node 5 → Node 6) is infeasible. If $x < y$ is true, the code hits break at Node 5 and immediately exits the entire while loop, so the for loop (Node 6) is never reached within that same iteration.

5.
To satisfy all-def coverage, each variable's definition must be exercised at least once.
Test Case 1:

- Input: y = 10, x = y
- Execution Path: 1 → 2 → 3 (False) → 6 → 7 → 6 → 8 → 9 → 2 → 11

Test Case 2: Break condition triggered

- Input: y = 2, x = y
- Execution Path: 1 → 2 → 3 (True) → 4 → 5 → 11

6.

To satisfy all-use coverage, every DU pair must be executed at least once.
Test Cases:

- y = 3, x = 5 (Triggers if (x < y), enters loop)
- y = 6, x = 100 (Skips while)
- y = 2, x = 1 (Takes else branch)

7.
To satisfy all-du-paths coverage, all DU pairs must be covered along every possible path.
Test Cases:

- y = 3, x = 5 (Covers x in loop)
- y = 6, x = 100 (Directly prints without entering loop)
- y = 2, x = 1 (Tests else condition)
- y = 1, x = 1 (Ensures loop runs and exits)

Q3:
1.
INVALID:
(s1 <= 0 || s2 <= 0 || s3 <= 0) || (s1 + s2 <= s3 || s2 + s3 <= s1 || s1 + s3 <= s2)

EQUILATERAL:
(s1 > 0 && s2 > 0 && s3 > 0) && (s1 + s2 > s3 && s2 + s3 > s1 && s1 + s3 > s2) && (s1 == s2 && s2 == s3)

ISOSCELES:
(s1 > 0 && s2 > 0 && s3 > 0) && (s1 + s2 > s3 && s2 + s3 > s1 && s1 + s3 > s2) && !(s1 == s2 && s2 == s3) && (s1 == s2 || s2 == s3 || s1 == s3)

SCALENE:
(s1 > 0 && s2 > 0 && s3 > 0) && (s1 + s2 > s3 && s2 + s3 > s1 && s1 + s3 > s2) && !(s1 == s2 && s2 == s3) && !(s1 == s2 || s2 == s3 || s1 == s3)

Line 7: $s1 > 0 \land s2 > 0 \land s3 > 0$
Line 10: $s1 > 0 \land s2 > 0 \land s3 > 0 \land (s1 + s2 > s3) \land (s2 + s3 > s1) \land (s1 + s3 > s2)$
Line 13: $s1 > 0 \land s2 > 0 \land s3 > 0 \land (s1 + s2 > s3) \land (s2 + s3 > s1) \land (s1 + s3 > s2) \land (s1 = s2 = s3)$
Line 16: $s1 > 0 \land s2 > 0 \land s3 > 0 \land (s1 + s2 > s3) \land (s2 + s3 > s1) \land (s1 + s3 > s2) \land \neg(s1 = s2 = s3) \land ((s1 = s2) \lor (s2 = s3) \lor (s1 = s3))$
Line 17: $s1 > 0 \land s2 > 0 \land s3 > 0 \land (s1 + s2 > s3) \land (s2 + s3 > s1) \land (s1 + s3 > s2) \land (s1 \neq s2) \land (s2 \neq s3) \land (s1 \neq s3)$

2.
TC1: (0, 1, 1) → invalid (non-positive sides)

TC2: (1, 1, 3) → invalid (triangle inequality fails)
TC3: (2, 2, 2) → equilateral
TC4: (5, 5, 3) → isosceles
TC5: (3, 4, 5) → scalene

3.
Conditions:
s1 <= 0,
s2 <= 0,
s3 <= 0,
s1 + s2 <= s3,
s2 + s3 <= s1,
s1 + s3 <= s2,
s1 == s2,
s2 == s3,
s1 == s3

Test Cases:
(0, 1, 1): s1 <= 0 true → INVALID
(1, 0, 1): s2 <= 0 true → INVALID
(1, 1, 0): s3 <= 0 true → INVALID
(1, 1, 3): s1 + s2 <= s3 true → INVALID
(3, 1, 1): s2 + s3 <= s1 true → INVALID
(1, 3, 1): s1 + s3 <= s2 true → INVALID
(2, 2, 2): s1 == s2 && s2 == s3 true → EQUILATERAL
(2, 2, 3): s1 == s2 true, s2 == s3 false → ISOSCELES
(2, 3, 2): s2 == s3 false, s1 == s3 true → ISOSCELES
(3, 4, 5): All equality conditions false → SCALENE

4.
This assumes the triangle is valid; otherwise, INVALID takes precedence.
INVALID: (s1 <= 0 || s2 <= 0 || s3 <= 0) || (s1 + s2 <= s3 || s2 + s3 <= s1 || s1 + s3 <= s2)
EQUILATERAL: (s1 == s2) && (s2 == s3)
ISOSCELES: (s1 == s2 || s2 == s3 || s1 == s3) && !(s1 == s2 && s2 == s3)
SCALENE: !(s1 == s2 || s2 == s3 || s1 == s3)

5.
Predicate 1 (INVALID): (s1 <= 0 || s2 <= 0 || s3 <= 0) || (s1 + s2 <= s3 || s2 + s3 <= s1 || s1 + s3 <= s2)
Test cases: (0, 1, 1), (1, 0, 1), (1, 1, 0), (1, 1, 3), (3, 1, 1), (1, 3, 1)

Predicate 2 (EQUILATERAL): s1 == s2 && s2 == s3
Test cases: (2, 2, 2) (true), (2, 2, 3) (false for s2 == s3)

Predicate 3 (ISOSCELES): s1 == s2 || s2 == s3 || s1 == s3
Test cases: (2, 2, 3), (2, 3, 2), (3, 2, 2), (3, 4, 5) (false)

Scalene: Automatically covered when no sides are equal: (3, 4, 5)

6.
All paths and conditions are reachable:
A triangle can be invalid (e.g., 0, 1, 1 or 1, 1, 3).
A triangle can be equilateral (e.g., 2, 2, 2).
A triangle can be isosceles (e.g., 2, 2, 3).
A triangle can be scalene (e.g., 3, 4, 5).

Potential Infeasible Requirements:

- Simultaneous violation of multiple constraints
- Extreme side length combinations that logically contradict each other
- Numerical limits that create unreachable conditions

Example of Infeasible Requirement:

- A test case where (sl + s2 ≤ s3) AND (sl == s2 == s3)