04. Pseudocode Solved PYQs & Comprehensive Practice Bank

> **Module Focus**: 20 Questions in 20 Minutes
> **Key Skills**: Bitwise Operator Tracing, Loop Execution Limits, Pass-by-Reference/Value, Recursion Call Stack Unwinding, Array Bit Manipulation, Operator Precedence

---

1. Bitwise Operations PYQ Bank

PYQ 1: Bitwise XOR and AND Combination

```
Integer a, b, c
Set a = 5, b = 10, c = 2
If ((a ^ b ^ c) > (a & b & c))
    a = b ^ c
    b = a + c
Else
    c = a & b
    a = b + c
End If
Print a + b + c
```

Step-by-Step Execution Trace: 1. **Binary Representation**: - $a = 5 = 0101_2$ - $b = 10 = 1010_2$ - $c = 2 = 0010_2$ 2. **Evaluate Condition** `(a ^ b ^ c) > (a & b & c)`: - $a \oplus b = 0101_2 \oplus 1010_2 = 1111_2 = 15$. - $15 \oplus c = 1111_2 \oplus 0010_2 = 1101_2 = 13$. Left side $= 13$. - $a \ \& \ b = 0101_2 \ \& \ 1010_2 = 0000_2 = 0$. - $0 \ \& \ c = 0$. Right side $= 0$. - Condition `13 > 0` is **TRUE**! 3. **Execute THEN block**: - `a = b ^ c` $\implies 10 \oplus 2 = 1010_2 \oplus 0010_2 = 1000_2 = 8$. (Now $a = 8$). - `b = a + c` $\implies 8 + 2 = 10$. (Now $b = 10$). 4. **Compute Final Expression** `a + b + c`: - $a = 8, b = 10, c = 2$. - $8 + 10 + 2 = 20$.

Final Output: `20`

PYQ 2: Bit Shift Arithmetic Combined with Logical Negation

```
Integer p = 12, q = 3, r = 0
r = (p >> 2) + (q << 2)
If ((r & 1) == 0)
    p = r ^ q
Else
    q = r & p
```

```
End If
Print p + q + r
```

Step-by-Step Execution Trace: 1. **Bit Shifts**: - `p >> 2` $\implies 12 \gg 2 = \lfloor 12 / 2^2 \rfloor = \lfloor 12 / 4 \rfloor = 3$. - `q << 2` $\implies 3 \ll 2 = 3 \times 2^2 = 3 \times 4 = 12$. - `r = 3 + 12 = 15`. 2. **Condition Check `(r & 1) == 0`**: - $15 \ \& \ 1 = 1111_2 \ \& \ 0001_2 = 1$. - `(1 == 0)` evaluates to **FALSE**. 3. **Execute ELSE block**: - `q = r & p` $\implies 15 \ \& \ 12 = 1111_2 \ \& \ 1100_2 = 1100_2 = 12$. (Now $q = 12$). 4. **Compute Final Sum `p + q + r`**: - $p = 12, q = 12, r = 15$. - $12 + 12 + 15 = 39$.

Final Output: `39`

PYQ 3: Complex Nested Bitwise & Subtraction Loop

```
Integer a = 14, b = 7, c = 0
While (a > b)
    c = c + (a & b)
    a = a - 2
    b = b + 1
End While
Print c
```

Step-by-Step Execution Trace Table:

Iteration	Initial `a`	Initial `b`	Condition `(a > b)`	`a & b` Calculation	Updated `c`	Updated `a`	Updated `b`
Start	14	7	-	-	0	14	7
Pass 1	14	7	$14 > 7$ (True)	$1110_2 \ \& \ 0111_2 = 0110_2 = 6$	$0 + 6 = 6$	$14 - 2 = 12$	$7 + 1 = 8$
Pass 2	12	8	$12 > 8$ (True)	$1100_2 \ \& \ 1000_2 = 1000_2 = 8$	$6 + 8 = 14$	$12 - 2 = 10$	$8 + 1 = 9$
Pass 3	10	9	$10 > 9$ (True)	$1010_2 \ \& \ 1001_2 = 1000_2 = 8$	$14 + 8 = 22$	$10 - 2 = 8$	$9 + 1 = 10$
Pass 4	8	10	$8 > 10$ (FALSE)	Loop Terminates	22	8	10

Final Output: `22`

PYQ 4: Bitwise OR, XOR, AND with Shift Operators

```
Integer a = 8, b = 5, c = 3
c = (a >> 1) ^ (b << 1)
If ((c | a) > (c & b))
    a = (a ^ b) + c
```

```
    b = b & c
Else
    c = c ^ a
End If
Print a + b + c
```

Step-by-Step Execution Trace: 1. **Bit Shift and Initial XOR Assignment**: - `a >> 1` $\implies 8 \gg 1 = 4 = 0100_2$. - `b << 1` $\implies 5 \ll 1 = 10 = 1010_2$. - `c = 4 ^ 10` $\implies 0100_2 \oplus 1010_2 = 1110_2 = 14$. 2. **Evaluate Condition** `(c | a) > (c & b)`: - `c | a` $\implies 14 \mid 8 = 1110_2 \mid 1000_2 = 1110_2 = 14$. - `c & b` $\implies 14 \ \& \ 5 = 1110_2 \ \& \ 0101_2 = 0100_2 = 4$. - Condition `14 > 4` is **TRUE**. 3. **Execute THEN block**: - `a = (a ^ b) + c` $\implies (8 \oplus 5) + 14 = (1000_2 \oplus 0101_2) + 14 = 13 + 14 = 27$. (Now $a = 27$). - `b = b & c` $\implies 5 \ \& \ 14 = 0101_2 \ \& \ 1110_2 = 0100_2 = 4$. (Now $b = 4$). 4. **Compute Final Expression** `a + b + c`: - $a = 27, b = 4, c = 14$. - $27 + 4 + 14 = 45$.

Final Output: `45`

PYQ 5: Bitwise Accumulation Loop with AND & OR

```
Integer a = 7, b = 11, c = 0
For (Integer i = 1 to 3)
    c = c + ((a & i) ^ (b | i))
End For
Print c
```

Step-by-Step Execution Trace Table:

Iteration `i`	`a & i` Binary & Value	`b | i` Binary & Value	`(a & i) ^ (b | i)` Calculation	Updated `c`
Start	-	-	-	0
i = 1	$0111_2 \ \& \ 0001_2 = 1$	$1011_2 \mid 0001_2 = 1011_2 = 11$	$0001_2 \oplus 1011_2 = 1010_2 = 10$	$0 + 10 = 10$
i = 2	$0111_2 \ \& \ 0010_2 = 2$	$1011_2 \mid 0010_2 = 1011_2 = 11$	$0010_2 \oplus 1011_2 = 1001_2 = 9$	$10 + 9 = 19$
i = 3	$0111_2 \ \& \ 0011_2 = 3$	$1011_2 \mid 0011_2 = 1011_2 = 11$	$0011_2 \oplus 1011_2 = 1000_2 = 8$	$19 + 8 = 27$

Final Output: `27`

---

2. Recursion & Stack Unwinding PYQ Bank

PYQ 6: Double Recursive Call Tree Tracing

```
Integer fun(Integer n)
    If (n <= 0)
        Return 0
    End If
    If (n == 1)
        Return 2
    End If
    Return fun(n - 1) + fun(n - 2) + n
End Function
What is the return value of fun(4)?
```

Step-by-Step Stack Unwinding Trace: 1. Base Cases: - `fun(0) = 0` - `fun(1) = 2` 2. Evaluate `fun(2)`: - `fun(2) = fun(1) + fun(0) + 2 = 2 + 0 + 2 = 4`. 3. Evaluate `fun(3)`: - `fun(3) = fun(2) + fun(1) + 3 = 4 + 2 + 3 = 9`. 4. Evaluate `fun(4)`: - `fun(4) = fun(3) + fun(2) + 4 = 9 + 4 + 4 = 17`.

Final Output: **`17`**

PYQ 7: Recursive Function with Static Variable State

```
Integer fun(Integer n)
    Static Integer count = 0
    If (n <= 0)
        Return count
    End If
    count = count + n
    Return fun(n - 1) + fun(n - 2)
End Function
What is the value returned by fun(3)?
```

Step-by-Step Call Order & State Unwinding: 1. `fun(3)`: `count` updated $\implies 0 + 3 = 3$. Calls `fun(2)` + `fun(1)`. 2. Left Child `fun(2)` executes first: - `count` updated $\implies 3 + 2 = 5$. Calls `fun(1)` + `fun(0)`. - Left Child `fun(1)` executes: - `count` updated $\implies 5 + 1 = 6$. Calls `fun(0)` + `fun(-1)`. - `fun(0)` returns current `count` $= 6$. - `fun(-1)` returns current `count` $= 6$. - `fun(1)` unwinds $\implies 6 + 6 = 12$. - Right Child `fun(0)` of `fun(2)` executes: - Returns current `count` $= 6$. - `fun(2)` unwinds $\implies 12 + 6 = 18$. 3. Right Child `fun(1)` of `fun(3)` executes: - `count` updated $\implies 6 + 1 = 7$. Calls `fun(0)` + `fun(-1)`. - `fun(0)` returns current `count` $= 7$. - `fun(-1)` returns current `count` $= 7$. - `fun(1)` unwinds $\implies 7 + 7 = 14$. 4. Root `fun(3)` unwinds: - `fun(3) = fun(2) + fun(1) = 18 + 14 = 32`.

Final Output: **`32`**

PYQ 8: Recursive Function with Bitwise XOR & Unwinding Step

```
Integer foo(Integer a, Integer b)
    If (b <= 0)
        Return a
    End If
    Return foo(a ^ b, b - 1) + b
End Function
What is the output of foo(5, 3)?
```

Step-by-Step Call Stack Recursion Trace: - **Call 1**: `foo(5, 3)` - $b = 3 > 0$. Bitwise XOR: $a \oplus b = 5 \oplus 3 = 0101_2 \oplus 0011_2 = 0110_2 = 6$. - Returns `foo(6, 2) + 3`. - **Call 2**: `foo(6, 2)` - $b = 2 > 0$. Bitwise XOR: $a \oplus b = 6 \oplus 2 = 0110_2 \oplus 0010_2 = 0100_2 = 4$. - Returns `foo(4, 1) + 2`. - **Call 3**: `foo(4, 1)` - $b = 1 > 0$. Bitwise XOR: $a \oplus b = 4 \oplus 1 = 0100_2 \oplus 0001_2 = 0101_2 = 5$. - Returns `foo(5, 0) + 1`. - **Call 4**: `foo(5, 0)` - $b = 0 \le 0$. **Base Case Reached**! Returns $a = 5$.

Stack Unwinding: - `foo(5, 0) = 5` - `foo(4, 1) = 5 + 1 = 6` - `foo(6, 2) = 6 + 2 = 8` - `foo(5, 3) = 8 + 3 = 11`

Final Output: **`11`**

PYQ 9: Alternating Parity Recursive Function

```
Integer solve(Integer n, Integer k)
    If (n <= 1)
        Return k
    End If
    If (n Mod 2 == 0)
        Return solve(n / 2, k + n)
    Else
        Return solve(n - 1, k * 2)
    End If
End Function
What is the return value of solve(6, 2)?
```

Step-by-Step Tail-Recursive Execution: 1. `solve(6, 2)`: $n=6$ (Even) $\implies$ Returns `solve(6 / 2, 2 + 6)` $=$ `solve(3, 8)`. 2. `solve(3, 8)`: $n=3$ (Odd) $\implies$ Returns `solve(3 - 1, 8 * 2)` $=$ `solve(2, 16)`. 3. `solve(2, 16)`: $n=2$ (Even) $\implies$ Returns `solve(2 / 2, 16 + 2)` $=$ `solve(1, 18)`. 4. `solve(1, 18)`: $n=1 \le 1$ (Base Case) $\implies$ Returns $k = 18$.

Final Output: **`18`**

---

3. Loop Execution & State Table PYQ Bank

PYQ 10: REPEAT-UNTIL Off-by-One Counter Verification

```
Integer i = 1, sum = 0
REPEAT
    sum = sum + i * i
    i = i + 1
UNTIL (i > 4)
Print sum
```

Step-by-Step Execution Trace: - **Pass 1**: `sum = 0 + (1 * 1) = 1`, `i = 2`. Condition `(2 > 4)` is **FALSE** $\to$ Continue. - **Pass 2**: `sum = 1 + (2 * 2) = 5`, `i = 3`. Condition `(3 > 4)` is **FALSE** $\to$ Continue. - **Pass 3**: `sum = 5 + (3 * 3) = 14`, `i = 4`. Condition `(4 > 4)` is **FALSE** $\to$ Continue. - **Pass 4**: `sum = 14 + (4 * 4) = 30`, `i = 5`. Condition `(5 > 4)` is **TRUE** $\to$ **TERMINATE**!

Final Output: `30` (Computes sum of squares $1^2 + 2^2 + 3^2 + 4^2 = 1 + 4 + 9 + 16 = 30$).

PYQ 11: Nested For Loops with Conditional Bitwise Accumulation

```
Integer sum = 0
For (Integer i = 1 to 3)
    For (Integer j = i to 3)
        If ((i + j) Mod 2 == 0)
            sum = sum + (i ^ j)
        Else
            sum = sum + (i & j)
        End If
    End For
End For
Print sum
```

Step-by-Step Execution Trace Table:

Outer `i`	Inner `j`	`i + j`	Parity	Evaluated Operation	Operation Result	Updated `sum`
1	1	2	Even	$1 \oplus 1 = 0001_2 \oplus 0001_2$	0	$0 + 0 = 0$
1	2	3	Odd	$1 \ \& \ 2 = 0001_2 \ \& \ 0010_2$	0	$0 + 0 = 0$
1	3	4	Even	$1 \oplus 3 = 0001_2 \oplus 0011_2$	2	$0 + 2 = 2$
2	2	4	Even		0	$2 + 0 = 2$

Outer `i`	Inner `j`	`i + j`	Parity	Evaluated Operation	Operation Result	Updated `sum`
				$2 \oplus 2 = 0010_2 \oplus 0010_2$		
2	3	5	Odd	$2 \ \& \ 3 = 0010_2 \ \& \ 0011_2$	2	$2 + 2 = 4$
3	3	6	Even	$3 \oplus 3 = 0011_2 \oplus 0011_2$	0	$4 + 0 = 4$

Final Output: `4`

PYQ 12: Digit Extraction Loop with Parity Branching

```
Integer n = 456, sum = 0, rem = 0
While (n > 0)
    rem = n Mod 10
    If (rem Mod 2 != 0)
        sum = sum + rem * 2
    Else
        sum = sum + rem
    End If
    n = n / 10
End While
Print sum
```

Step-by-Step Execution Trace Table:

Iteration	Initial `n`	`rem = n Mod 10`	`rem Mod 2 != 0`	Applied Operation	Updated `sum`	`n = n / 10`
Pass 1	456	6	False (Even)	`sum + rem`	$0 + 6 = 6$	45
Pass 2	45	5	True (Odd)	`sum + rem * 2`	$6 + (5 \times 2) = 16$	4
Pass 3	4	4	False (Even)	`sum + rem`	$16 + 4 = 20$	0
Pass 4	0	-	$0 > 0$ False	Loop Terminates	20	0

Final Output: `20`

PYQ 13: Bitwise Right Shift & XOR Accumulator Loop

```
Integer x = 15, y = 9, z = 0
While (x > 0)
    z = z + (x & 1)
```

```
    x = x >> 1
    y = y ^ x
End While
Print z, y
```

Step-by-Step Execution Trace Table:

Iteration	Initial `x`	`x & 1`	Updated `z`	`x >> 1` (New `x`)	`y ^ x` Calculation	Updated `y`
Start	15 (1111_2)	-	0	-	-	9 (1001_2)
Pass 1	15	1	$0 + 1 = 1$	7 (0111_2)	$9 \oplus 7 = 1001_2 \oplus 0111_2 = 1110_2$	14
Pass 2	7	1	$1 + 1 = 2$	3 (0011_2)	$14 \oplus 3 = 1110_2 \oplus 0011_2 = 1101_2$	13
Pass 3	3	1	$2 + 1 = 3$	1 (0001_2)	$13 \oplus 1 = 1101_2 \oplus 0001_2 = 1100_2$	12
Pass 4	1	1	$3 + 1 = 4$	0 (0000_2)	$12 \oplus 0 = 1100_2 \oplus 0000_2 = 1100_2$	12
Pass 5	0	-	Loop Ends	0	-	12

Final Output: `4 12`

---

4. Pass-by-Reference & Array Tracing PYQ Bank

PYQ 14: Pass-by-Reference Modifying Outer Variables

```
Function modify(Integer &x, Integer y)
    x = x * 2
    y = y + 5
    Return x + y
End Function

Main:
    Integer a = 5, b = 10, res
    res = modify(a, b)
    Print a, b, res
End Main
```

Step-by-Step Execution Trace: 1. `a` is passed **by reference** (`&x`), so changes to `x` directly mutate variable `a`. 2. `b` is passed **by value** (`y`), so `y` is a local copy; variable `b` remains unaffected. 3. Inside `modify`: - `x = x * 2` $\implies x = 5 \times 2 = 10$. Variable `a` becomes `10`. - `y = y + 5` $\implies y = 10 + 5 = 15$. Variable `b` stays `10`. - Returns `10 + 15 = 25`. 4. Printing `a, b, res`: - `a = 10`, `b = 10`, `res = 25`.

Final Output: `10 10 25`

PYQ 15: Pass-by-Reference Bitwise Swap and Sum

```
Function update(Integer &a, Integer b, Integer &c)
    a = a ^ b
    b = b ^ a
    c = a + b
    Return a * b
End Function

Main:
    Integer x = 6, y = 4, z = 2, ans
    ans = update(x, y, z)
    Print x, y, z, ans
End Main
```

Step-by-Step Parameter Mutation Trace: 1. `x` is passed by reference (`&a`) $\implies$ modifications to `a` directly alter outer `x`. 2. `y` is passed by value (`b`) $\implies$ modifications to `b` do NOT affect outer `y`. 3. `z` is passed by reference (`&c`) $\implies$ modifications to `c` directly alter outer `z`. 4. Inside `update`: - `a = a ^ b` $\implies 6 \oplus 4 = 0110_2 \oplus 0100_2 = 0010_2 = 2$. (Outer `x` becomes **`2`**). - `b = b ^ a` $\implies 4 \oplus 2 = 0100_2 \oplus 0010_2 = 0110_2 = 6$. (Local `b` $= 6$; outer `y` stays **`4`**). - `c = a + b` $\implies 2 + 6 = 8$. (Outer `z` becomes **`8`**). - `Return a * b` $\implies 2 \times 6 = 12$. 5. In `Main`: - `x = 2`, `y = 4`, `z = 8`, `ans = 12`.

Final Output: `2 4 8 12`

PYQ 16: Array Traversal with Bit Shift & Conditional Accumulation

```
Integer A[5] = {4, 7, 12, 15, 8}
Integer total = 0
For (Integer i = 0 to 4)
    If ((A[i] & 1) == 0)
        total = total + (A[i] >> 1)
    Else
        total = total + (A[i] << 1)
    End If
End For
Print total
```

Step-by-Step Execution Trace Table:

Index `i`	Array Element `A[i]`	`A[i] & 1` Check	Parity Branch	Shift Operation & Value	Updated `total`
0	4	$4 \ \& \ 1 = 0$			$0 + 2 = 2$

Index `i`	Array Element `A[i]`	`A[i] & 1` Check	Parity Branch	Shift Operation & Value	Updated `total`
			Even (`== 0`)	$4 \gg 1 = \lfloor 4/2 \rfloor = 2$	
1	7	$7 \& 1 = 1$	Odd (`!= 0`)	$7 \ll 1 = 7 \times 2 = 14$	$2 + 14 = 16$
2	12	$12 \& 1 = 0$	Even (`== 0`)	$12 \gg 1 = \lfloor 12/2 \rfloor = 6$	$16 + 6 = 22$
3	15	$15 \& 1 = 1$	Odd (`!= 0`)	$15 \ll 1 = 15 \times 2 = 30$	$22 + 30 = 52$
4	8	$8 \& 1 = 0$	Even (`== 0`)	$8 \gg 1 = \lfloor 8/2 \rfloor = 4$	$52 + 4 = 56$

Final Output: `56`

---

5. Operator Precedence & Conditional Logic Bank

PYQ 17: Bitwise AND vs XOR Operator Precedence Tracing

```
Integer p = 5, q = 3, r = 8
Integer res = (p + q * 2) ^ (r >> 2) & 7
Print res
```

Step-by-Step Precedence & Binary Evaluation: 1. **Parentheses & Arithmetic**: - `(p + q * 2)` $\implies 5 + (3 \times 2) = 5 + 6 = 11 = 1011_2$. - `(r >> 2)` $\implies 8 \gg 2 = 2 = 0010_2$. - Expression reduces to: `11 ^ 2 & 7`. 2. **Bitwise Operator Precedence (`&` higher than `^`)**: - Evaluate `2 & 7` first: $0010_2 \& 0111_2 = 0010_2 = 2$. 3. **Evaluate Bitwise XOR (`^`)**: - `11 ^ 2` $\implies 1011_2 \oplus 0010_2 = 1001_2 = 9$.

Final Output: `9`

PYQ 18: Ternary Operator Evaluation with Relational Precedence

```
Integer x = 10, y = 20, z = 15, ans
ans = (x ^ y > z) ? ((y >> 2) + x) : ((x << 1) ^ z)
Print ans
```

Step-by-Step Precedence & Execution: 1. **Evaluate Condition** `(x ^ y > z)` : - Relational operator `>` has higher precedence than bitwise `^`. - `y > z` $\implies 20 > 15 \implies 1$ (True). - Expression becomes `x ^ 1` $\implies 10 \oplus 1 = 1010_2 \oplus 0001_2 = 1011_2 = 11$. - Non-zero integer `11` evaluates to **TRUE** in

boolean condition context. 2. **Execute TRUE Expression** `((y >> 2) + x)`: - `y >> 2` $\implies 20 \gg 2 = 5$. - `5 + x` $\implies 5 + 10 = 15$. 3. **Assign & Print**: - `ans = 15`.

Final Output: `15`

---

6. Web-Researched CoCubes, Sanfoundry & GFG High-Frequency PYQ Expansion

PYQ 19: CoCubes Bitwise XOR Inequality Branching Trap

```
Integer a = 6, b = 8, c = 15
If ((b ^ a) < a)
    b = a
End If
Print a + b + c
```

Step-by-Step Binary Execution Trace: 1. **Binary Conversions**: - $a = 6 = 0110_2$ - $b = 8 = 1000_2$ - $c = 15 = 1111_2$ 2. **Evaluate Condition** `(b ^ a) < a`: - $b \oplus a = 1000_2 \oplus 0110_2 = 1110_2 = 14$. - Evaluate `14 < 6` $\implies$ **FALSE**. 3. **Branch Decision**: - The `If` condition fails, so `b = a` is **skipped**. `b` remains `8`. 4. **Compute Final Output** `a + b + c`: - $6 + 8 + 15 = 29$.

State Table Summary: | Operation | Left Side | Right Side | Condition | Action Taken | Final Value | | :---: | :---: | :---: | :---: | :---: | :---: | | `b ^ a` | $1000_2 \oplus 0110_2 = 14$ | $a = 6$ | `14 < 6` (FALSE) | Skip `b = a` | `b = 8` | | `a + b + c` | $6 + 8 + 15$ | - | - | Print Output | `29` |

Final Output: `29`

PYQ 20: CoCubes Recursive Parameter Swapper & Accumulator

```
Function solve(Integer a, Integer b)
    If (a < b)
        Return solve(b, a)
    Else If (b != 0)
        Return (a + solve(a, b - 1))
    Else
        Return 0
    End If
End Function

What is the returned value of solve(8, 9)?
```

Step-by-Step Recursive Call Stack Unwinding Table: 1. **Call 1**: `solve(8, 9)` $\implies (8 < 9)$ is **TRUE**. Returns `solve(9, 8)` (Parameter order swapped so $a \ge b$). 2. **Call 2 to Call 10 Winding Phase**: `solve(9, 8)` calls `9 + solve(9, 7)` down to `solve(9, 0)`.

Call Level	Invocation	Condition / Action	Returned Value / Expression	Accumulated Value
Level 10	`solve(9, 0)`	Base Case `b == 0`	Returns `0`	**0**
Level 9	`solve(9, 1)`	`9 + solve(9, 0)`	$9 + 0$	**9**
Level 8	`solve(9, 2)`	`9 + solve(9, 1)`	$9 + 9$	**18**
Level 7	`solve(9, 3)`	`9 + solve(9, 2)`	$9 + 18$	**27**
Level 6	`solve(9, 4)`	`9 + solve(9, 3)`	$9 + 27$	**36**
Level 5	`solve(9, 5)`	`9 + solve(9, 4)`	$9 + 36$	**45**
Level 4	`solve(9, 6)`	`9 + solve(9, 5)`	$9 + 45$	**54**
Level 3	`solve(9, 7)`	`9 + solve(9, 6)`	$9 + 54$	**63**
Level 2	`solve(9, 8)`	`9 + solve(9, 7)`	$9 + 63$	**72**
Level 1	`solve(8, 9)`	Returns `solve(9, 8)`	Returns Level 2 result	`72`

Final Output: `72`

PYQ 21: CoCubes Decrement Modification Loop Trap

```
Integer n
For (n = 3; n != 0; n--)
    Print n
    n = n - 1
End For
```

Step-by-Step State Table Trace: | Iteration | Initial `n` | Condition `n != 0` | Body Action (`Print n`) | Internal Mutation (`n = n - 1`) | Post Loop Decrement (`n--`) | Value for Next Test | | :---: | :---: | :---: | :---: | :---: | :---: | :---: | | **1** | 3 | `3 != 0` (**TRUE**) | Prints `3` | `n = 3 - 1 = 2` | `n = 2 - 1 = 1` | **1** | | **2** | 1 | `1 != 0` (**TRUE**) | Prints `1` | `n = 1 - 1 = 0` | `n = 0 - 1 = -1` | **-1** | | **3** | -1 | `-1 != 0` (**TRUE**) | Prints `-1` | `n = -1 - 1 = -2` | `n = -2 - 1 = -3` | **-3** | | **4** | -3 | `-3 != 0` (**TRUE**) | Prints `-3` | `n = -3 - 1 = -4` | `n = -4 - 1 = -5` | **-5** |

- **Key Observation**: Notice how `n` skips `0` completely! It transitions from `1` $\to$ `0` inside body $\to$ `-1` after loop header `n--`.
- **Termination Check**: `n != 0` will remain **TRUE** for all negative odd integers.

Final Output: **Infinite Loop**

PYQ 22: Sanfoundry REPEAT-UNTIL Shift & Mask Counter

```
Integer p = 10, q = 3, count = 0
REPEAT
```

```
    count = count + 1
    p = p >> 1
    q = q + (p & 1)
UNTIL (p <= 1)
Print count, p, q
```

Step-by-Step Variable State Table: | Pass | Initial `p` | `count = count + 1` | `p = p >> 1` | `p & 1` | `q = q + (p & 1)` | UNTIL Test `(p <= 1)` | Evaluation | | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | | **Start** | 10 | 0 | - | - | 3 | - | - | | **Pass 1** | 10 | **1** | $10 \gg 1 = \mathbf{5}$ (`0101`) | $5 \ \& \ 1 = \mathbf{1}$ | $3 + 1 = \mathbf{4}$ | `(5 <= 1)` | **FALSE** $\to$ Continue | | **Pass 2** | 5 | **2** | $5 \gg 1 = \mathbf{2}$ (`0010`) | $2 \ \& \ 1 = \mathbf{0}$ | $4 + 0 = \mathbf{4}$ | `(2 <= 1)` | **FALSE** $\to$ Continue | | **Pass 3** | 2 | **3** | $2 \gg 1 = \mathbf{1}$ (`0001`) | $1 \ \& \ 1 = \mathbf{1}$ | $4 + 1 = \mathbf{5}$ | `(1 <= 1)` | **TRUE** $\to$ **TERMINATE** |

- **Final Values**: `count = 3`, `p = 1`, `q = 5`.

Final Output: `3 1 5`

PYQ 23: GFG Bitwise Left Shift with Two's Complement NOT Operator

```
Integer a = 7, b = 2, c = 0
c = ((~a) << b) + (a ^ b)
Print c
```

Step-by-Step Binary State Table: | Step | Operation / Sub-expression | 8-Bit Binary Computation | Decimal Result | Notes | | :---: | :--- | :--- | :---: | :--- | | **1** | `a` | `0000 0111` | **7** | Input Value | | **2** | `~a` | `1111 1000` | **-8** | Bitwise NOT identity $\sim a = -(a + 1) = -8$ | | **3** | `(~a) << b` | $11111000_2 \ll 2 = \mathbf{1110\ 0000}_2$ | **-32** | Shift left by 2 ($(-8) \times 4 = -32$) | | **4** | `a ^ b` | $00000111_2 \oplus 00000010_2 = \mathbf{0000\ 0101}_2$ | **5** | Bitwise XOR of 7 and 2 | | **5** | `c = (-32) + 5` | $-32 + 5$ | **-27** | Addition of components |

Final Output: `-27`

💡 Quick Summary Checklist for Exam Mastery

1. **Bitwise Shifts**:
2. Left Shift ($x \ll n$) $= x \times 2^n$
3. Right Shift ($x \gg n$) $= \lfloor x / 2^n \rfloor$
4. **Bitwise Parity Check**:
5. $(x \ \& \ 1) == 0 \implies x$ is **Even**
6. $(x \ \& \ 1) == 1 \implies x$ is **Odd**
7. **Bitwise Identity Properties**:
8. $x \oplus x = 0$
9. $x \oplus 0 = x$
10. $x \ \& \ x = x$
11. $x \mid x = x$
12. **Pass-by-Reference**:

13. Parameter prefixed with `&` modifies original calling variable.
14. Value parameter makes local copy, leaving outer variable intact.