🔰 Full Plan (Structured Format)

We will divide a total of 30 programs into three levels.

Level	**Programs**	**Description**
Beginner	10	Basic number, loop, and string programs
Intermediate	10	Array, sorting, and logic programs
Advanced	10	DSA-based problems like matrix, recursion, tree, etc.

Each program will have 4 versions.

▨ C Language

▨ Java

▨ Python

▨ DSA with Java (optimized or data-structure version)

🚀 Start with Batch 1: Beginner Level Programs (1–10)

1️⃣Even or Odd Number

2️⃣Find Largest of Three Numbers

3️⃣Sum of Natural Numbers (Loop + Recursion)

4️⃣Reverse a Number

5️⃣Palindrome Number/String

6️⃣Prime Number Check

7️⃣Factorial of a Number

8️⃣Fibonacci Series Generation

9️⃣Count Digits in a Number

10️⃣Swap Two Numbers (using temp & without temp)

▨ Intermediate Level Programs (10 Programs)

1. Armstrong Number Check – A number is Armstrong if sum of its digits powered by number of digits equals the number itself.

2. Reverse an Array – Reverse elements of an array.

3. Find the Second Largest Element in Array – Identify 2nd maximum number in an array.

4. Find GCD / LCM of Two Numbers – Using loops or Euclidean algorithm.

5. Array Rotation (Left / Right) – Rotate array elements by given positions.

6. Remove Duplicates from Array – Eliminate repeated elements.

7. Count Frequency of Each Element in Array – Count how many times each element occurs.

8. Check for Array Palindrome – Determine if array reads same forwards and backwards.

9. Sorting an Array – Implement Bubble Sort and Selection Sort.

10. Merge Two Arrays – Combine two arrays into one (sorted/unsorted).

▨ Advanced Level Programs List

1. Matrix Multiplication / Transpose / Diagonal Sum – 2D array operations.

2. Check Balanced Parentheses Using Stack – Validate expression parentheses.

3. Balanced Binary Tree (Height balanced) – Check if binary tree is height-balanced.

4. Find Cycle in Linked List – Detect loop using Floyd's cycle detection algorithm.

5. Lowest Common Ancestor in Binary Tree – Find LCA of two nodes.

6. Find Subarray with Given Sum (Sliding Window) – Continuous subarray sum problem.

7. Regular Expression Pattern Matching – Validate patterns using regex.

8. Huffman Coding Implementation (Character Frequency Compression) – Data compression algorithm.

9. N-Queens Problem (Backtracking) – Place N queens on NxN chessboard.

10. Trapping Rainwater Problem – Optimize water trapping between heights.

Batch 1: Beginner Level Programs (1–10) – Logic Explanation

1️⃣Even or Odd Number

Logic Flow:

1. Input number n.

2. Divide number by 2 and check remainder: n % 2.

3. If remainder is 0 → number is Even.

4. Else → number is Odd.

Pseudo Code:

```
START
Input n
IF n mod 2 == 0 THEN
   PRINT "Even"
ELSE
   PRINT "Odd"
END
```

Explanation:

Number is divisible by 2 → Even. Otherwise → Odd.

2�Find Largest of Three Numbers

Logic Flow:

1. Input three numbers: a, b, c.

2. Initialize max = a.

3. Compare b with max. If b > max → update max = b.

4. Compare c with max. If c > max → update max = c.

5. Print max.

Pseudo Code:

```
START
Input a, b, c
max = a
IF b > max THEN max = b
IF c > max THEN max = c
PRINT max
END
```

Explanation:

Simple comparison using conditional statements to find the largest.

3️⃣Sum of Natural Numbers (Loop + Recursion)

Loop Method Logic:

1. Input n.

2. Initialize sum = 0.

3. Loop from 1 to n, add each number to sum.

4. Print sum.

Recursion Method Logic:

1. Define function sum(n):

Base case: if n == 1 → return 1.

Recursive step: return n + sum(n-1).

2. Call sum(n) and print result.

Pseudo Code (Loop):

```
START
Input n
sum = 0
FOR i = 1 TO n
    sum = sum + i
PRINT sum
END
```

Pseudo Code (Recursion):

```
FUNCTION sum(n)
    IF n == 1 RETURN 1
    ELSE RETURN n + sum(n-1)
END FUNCTION

START
Input n
PRINT sum(n)
END
```

Explanation:

Loop → iterative addition. Recursion → function calls itself to add numbers down to 1.

4️⃣Reverse a Number

Logic Flow:

1. Input number n.

2. Initialize rev = 0.

3. While n > 0:

Get last digit: digit = n % 10.

Update reverse: rev = rev * 10 + digit.

Remove last digit: n = n / 10.

4. Print rev.

Pseudo Code:

```
START
Input n
rev = 0
WHILE n > 0
    digit = n % 10
    rev = rev * 10 + digit
    n = n / 10
PRINT rev
END
```

Explanation:
Extract digits one by one from the end and build reverse number.

5️⃣ Palindrome Number/String

Logic Flow:

1. Input number/string x.

2. For number → reverse it (use logic from program 4).
For string → reverse characters.

3. Compare original and reversed.

4. If equal → Palindrome. Else → Not Palindrome.

Pseudo Code:

```
START
Input x
reversed = REVERSE(x)
IF x == reversed THEN
   PRINT "Palindrome"
ELSE
   PRINT "Not Palindrome"
END
```

Explanation:
Palindrome means reading forward == reading backward.

6️⃣ Prime Number Check

Logic Flow:

1. Input number n.

2. If n <= 1 → Not Prime.

3. Loop i from 2 to n/2:

If n % i == 0 → Not Prime.

4. If no divisor found → Prime.

Pseudo Code:

```
START
Input n
IF n <= 1 THEN PRINT "Not Prime"
ELSE
    flag = True
    FOR i = 2 TO n/2
        IF n % i == 0 THEN flag = False; BREAK
    IF flag THEN PRINT "Prime" ELSE PRINT "Not Prime"
END
```

Explanation:

Prime → only divisible by 1 and itself. Check all numbers till n/2.

7️⃣Factorial of a Number

Logic Flow:

Iterative:

1. Input n.

2. Initialize fact = 1.

3. Loop i = 1 to n → multiply: fact = fact * i.

4. Print fact.

Recursive:

1. Function fact(n):

Base case: if n == 0 → return 1.

Else → return n * fact(n-1).

Pseudo Code (Iterative):

```
START
Input n
fact = 1
FOR i = 1 TO n
   fact = fact * i
PRINT fact
END
```

Explanation:
Factorial = product of all numbers from 1 to n.

8️⃣Fibonacci Series Generation

Logic Flow:

1. Input n (number of terms).

2. Initialize a = 0, b = 1.

3. Loop i = 1 to n:

Print a.

next = a + b

a = b, b = next

Pseudo Code:

```
START
Input n
a = 0, b = 1
FOR i = 1 TO n
    PRINT a
    next = a + b
    a = b
    b = next
END
```

Explanation:

Each term = sum of previous two terms.

9️⃣Count Digits in a Number

Logic Flow:

1. Input number n.

2. Initialize count = 0.

3. While n > 0:

n = n / 10

count = count + 1

4. Print count.

Pseudo Code:

```
START
Input n
count = 0
WHILE n > 0
    n = n / 10
    count = count + 1
PRINT count
END
```

Explanation:
Divide by 10 repeatedly → number of iterations = number of digits.

🔟 Swap Two Numbers (using temp & without temp)

Logic Flow:

Using Temp:

1. Input a and b.

2. temp = a

3. a = b

4. b = temp

Without Temp:

1. a = a + b

2. b = a - b

3. a = a - b

Pseudo Code (Without Temp):

```
START
Input a, b
a = a + b
b = a - b
a = a - b
PRINT a, b
END
```

Explanation:
Mathematical trick to swap without extra variable.

Batch 2: Intermediate Level Programs (11–20) – Logic Explanation

11□Armstrong Number Check

Logic Flow:

1. Input number n.

2. Count number of digits → digits.

3. Initialize sum = 0.

4. For each digit in n:

sum += (digit ^ digits)

5. If sum == n → Armstrong. Else → Not Armstrong.

Pseudo Code:

```
START
Input n
digits = COUNT_DIGITS(n)
```

```
sum = 0
temp = n
WHILE temp > 0
    digit = temp % 10
    sum = sum + digit^digits
    temp = temp / 10
IF sum == n THEN PRINT "Armstrong" ELSE PRINT "Not Armstrong"
END
```

Explanation:

Armstrong → sum of digits each raised to power of number of digits = original number.

12️⃣Reverse an Array

Logic Flow:

1. Input array arr of size n.

2. Loop from i=0 to n/2:

Swap arr[i] with arr[n-1-i].

3. Print reversed array.

Pseudo Code:

```
START
Input n
Input arr[n]
FOR i = 0 TO n/2 - 1
    temp = arr[i]
    arr[i] = arr[n-1-i]
    arr[n-1-i] = temp
PRINT arr
END
```

Explanation:

Swap symmetric elements from start and end → array reversed.

13⃣Find Second Largest Element in Array

Logic Flow:

1. Input array arr of size n.

2. Initialize max = arr[0], second = -∞.

3. Loop i = 1 to n-1:

If arr[i] > max → second = max, max = arr[i]

Else if arr[i] > second and arr[i] != max → second = arr[i]

4. Print second.

Pseudo Code:

```
START
Input n
Input arr[n]
max = arr[0]; second = -∞
FOR i = 1 TO n-1
    IF arr[i] > max THEN second = max; max = arr[i]
    ELSE IF arr[i] > second AND arr[i] != max THEN second = arr[i]
PRINT second
END
```

Explanation:

Keep track of largest and second largest simultaneously.

14️⃣Find GCD / LCM of Two Numbers

Logic Flow:

1. Input a and b.

2. GCD (Euclid’s algorithm):

While b != 0:

temp = b; b = a % b; a = temp

3. LCM = (original_a * original_b) / GCD

4. Print GCD and LCM.

Pseudo Code:

```
START
Input a, b
gcd = GCD(a, b)
lcm = (a * b) / gcd
PRINT gcd, lcm
END

FUNCTION GCD(a, b)
    IF b == 0 RETURN a
    ELSE RETURN GCD(b, a % b)
END FUNCTION
```

Explanation:

GCD → largest number dividing both. LCM → smallest multiple of both.

15️ Array Rotation (Left / Right)

Logic Flow (Left Rotate by d positions):

1. Input array arr[n], positions d.

2. Loop d times:

Store first element in temp.

Shift all elements left by 1.

Place temp at end.

3. Print rotated array.

Pseudo Code:

```
START
Input n, d
Input arr[n]
FOR i = 1 TO d
    temp = arr[0]
    FOR j = 0 TO n-2
        arr[j] = arr[j+1]
    arr[n-1] = temp
PRINT arr
END
```

Explanation:
Left rotation → elements move left, first elements wrap to end.

16□Remove Duplicates from Array

Logic Flow:

1. Input array arr[n].

2. Initialize empty array res[].

3. For each element x in arr:

If x not in res → append x.

4. Print res.

Pseudo Code:

```
START
Input n
Input arr[n]
res = empty array
FOR i = 0 TO n-1
    IF arr[i] NOT IN res THEN append arr[i] to res
PRINT res
```

END

Explanation:
Check each element → add to result only if not already present.

17⃣Count Frequency of Each Element in Array

Logic Flow:

1. Input array arr[n].

2. Initialize visited[n] = 0.

3. Loop i = 0 to n-1:

If visited[i] == 1 → skip

Count occurrences of arr[i] in rest of array, mark visited.

4. Print element frequency.

Pseudo Code:

```
START
Input n
Input arr[n]
Initialize visited[n] = 0
FOR i = 0 TO n-1
    IF visited[i] == 1 THEN CONTINUE
    count = 1
    FOR j = i+1 TO n-1
        IF arr[i] == arr[j] THEN count++; visited[j]=1
    PRINT arr[i], count
END
```

Explanation:
Track elements seen → count each unique element’s frequency.

18􏰀Check for Array Palindrome

Logic Flow:

1. Input array arr[n].

2. Loop i = 0 to n/2:

Compare arr[i] with arr[n-1-i].

If mismatch → Not Palindrome.

3. If no mismatch → Palindrome.

Pseudo Code:

```
START
Input n
Input arr[n]
flag = True
FOR i = 0 TO n/2 - 1
    IF arr[i] != arr[n-1-i] THEN flag = False; BREAK
IF flag THEN PRINT "Palindrome" ELSE PRINT "Not Palindrome"
END
```

Explanation:
Array is palindrome if symmetric elements are equal.

19️⃣Sorting an Array (Bubble / Selection Sort)

Logic Flow (Bubble Sort):

1. Input array arr[n].

2. Loop i = 0 to n-1:

Loop j = 0 to n-i-2:

If arr[j] > arr[j+1] → swap

3. Print sorted array.

Pseudo Code:

```
START
Input n
Input arr[n]
FOR i = 0 TO n-2
    FOR j = 0 TO n-2-i
        IF arr[j] > arr[j+1] THEN SWAP(arr[j], arr[j+1])
PRINT arr
END
```

Explanation:

Repeatedly swap adjacent elements if in wrong order → largest "bubbles" to end.

20️⃣Merge Two Arrays

Logic Flow:

1. Input array a[n] and b[m].

2. Initialize new array c[n+m].

3. Copy elements of a → c[0..n-1].

4. Copy elements of b → c[n..n+m-1].

5. Print c.

Pseudo Code:

```
START
Input n
Input a[n]
Input m
Input b[m]
Initialize c[n+m]
FOR i = 0 TO n-1: c[i] = a[i]
FOR i = 0 TO m-1: c[n+i] = b[i]
PRINT c
END
```

Explanation:

Merge → concatenate two arrays into one.

==
==
==

Batch 3: Advanced Level Programs (21–30) – Logic Explanation

21□Matrix Multiplication / Transpose / Diagonal Sum

a) Matrix Multiplication Logic:

1. Input two matrices A[n][m] and B[m][p].

2. Initialize result matrix C[n][p] with zeros.

3. Loop i = 0..n-1, j = 0..p-1:

For k = 0..m-1 → C[i][j] += A[i][k] * B[k][j]

4. Print C.

Pseudo Code:

```
START
Input n, m, p
Input A[n][m], B[m][p]
Initialize C[n][p] = 0
FOR i = 0 TO n-1
    FOR j = 0 TO p-1
        FOR k = 0 TO m-1
            C[i][j] += A[i][k] * B[k][j]
PRINT C
END
```

b) Matrix Transpose Logic:

1. For each element A[i][j] → set Transpose[j][i] = A[i][j]

c) Diagonal Sum Logic:

1. Sum main diagonal → sum of A[i][i]

2. Sum secondary diagonal → sum of A[i][n-1-i]

22�Check Balanced Parentheses Using Stack

Logic Flow:

1. Input string of parentheses.

2. Initialize empty stack.

3. Traverse each character:

If opening (→ push onto stack.

If closing) → pop from stack.

If stack empty when trying to pop → Unbalanced

4. After traversal → if stack empty → Balanced else Unbalanced

Pseudo Code:

```
START
Input string s
stack = empty
FOR ch IN s
    IF ch == '(' THEN push(stack, ch)
    ELSE IF ch == ')' THEN
        IF stack empty THEN PRINT "Unbalanced"; END
        ELSE pop(stack)
IF stack empty THEN PRINT "Balanced" ELSE PRINT "Unbalanced"
END
```

23�Balanced Binary Tree (Height Balanced)

Logic Flow:

1. Define recursive function isBalanced(node):

If node is NULL → return True

Compute left height = height(node.left)

Compute right height = height(node.right)

If |left - right| > 1 → return False

Else check recursively left & right subtrees

Pseudo Code:

```
FUNCTION height(node)
    IF node == NULL RETURN 0
    RETURN 1 + MAX(height(node.left), height(node.right))
```

```
END

FUNCTION isBalanced(node)
    IF node == NULL RETURN True
    leftHeight = height(node.left)
    rightHeight = height(node.right)
    IF ABS(leftHeight - rightHeight) > 1 THEN RETURN False
    RETURN isBalanced(node.left) AND isBalanced(node.right)
END
```

24�Find Cycle in Linked List

Logic Flow (Floyd’s Cycle Detection):

1. Initialize two pointers: slow, fast at head.

2. Loop:

slow = slow.next

fast = fast.next.next

If slow == fast → Cycle exists

3. If fast reaches NULL → No cycle

Pseudo Code:

```
START
slow = head
fast = head
WHILE fast != NULL AND fast.next != NULL
    slow = slow.next
    fast = fast.next.next
    IF slow == fast THEN PRINT "Cycle Exists"; END
PRINT "No Cycle"
END
```

25□Lowest Common Ancestor in Binary Tree

Logic Flow:

1. If root is NULL → return NULL

2. If root matches n1 or n2 → return root

3. Recur for left and right subtrees:

left = LCA(root.left, n1, n2)

right = LCA(root.right, n1, n2)

4. If left != NULL and right != NULL → root is LCA

5. Else → return left or right

Pseudo Code:

```
FUNCTION LCA(root, n1, n2)
    IF root == NULL RETURN NULL
    IF root.data == n1 OR root.data == n2 THEN RETURN root
    left = LCA(root.left, n1, n2)
    right = LCA(root.right, n1, n2)
    IF left != NULL AND right != NULL THEN RETURN root
    RETURN left IF left != NULL ELSE right
END
```

26�Find Subarray with Given Sum (Sliding Window)

Logic Flow (for positive numbers):

1. Initialize start = 0, curr_sum = arr[0]

2. Loop i = 1 to n:

While curr_sum > sum and start < i-1 → subtract arr[start], increment start

If curr_sum == sum → print start to i-1

Add arr[i] to curr_sum

Pseudo Code:

```
START
Input arr[n], target_sum
start = 0
curr_sum = arr[0]
FOR i = 1 TO n
    WHILE curr_sum > target_sum AND start < i-1
        curr_sum = curr_sum - arr[start]
        start++
    IF curr_sum == target_sum THEN PRINT start, i-1
    IF i < n THEN curr_sum += arr[i]
END
```

2?□Regular Expression Pattern Matching

Logic Flow:

1. Input string text and pattern

2. Use regex library to match pattern

3. If match → print "Pattern Found" else → "Not Found"

Pseudo Code:

```
START
Input text, pattern
IF regex_match(text, pattern) THEN PRINT "Pattern Found"
ELSE PRINT "Not Found"
END
```

28�Huffman Coding Implementation

Logic Flow:

1. Count frequency of each character

2. Create min-heap of nodes (character + frequency)

3. While heap has more than 1 node:

Extract two nodes with least frequency

Create new node with sum frequency → push back to heap

4. Traverse tree to assign binary codes to characters

Pseudo Code:

```
START
Input text
freq = count frequency of each character
heap = min-heap of freq nodes
WHILE heap.size > 1
    left = heap.pop()
    right = heap.pop()
    merged = Node(freq=left.freq+right.freq, left, right)
    heap.push(merged)
root = heap.pop()
ASSIGN_CODES(root, "")
END

FUNCTION ASSIGN_CODES(node, code)
    IF node is leaf THEN PRINT node.char, code
    ELSE
        ASSIGN_CODES(node.left, code+"0")
        ASSIGN_CODES(node.right, code+"1")
END
```

29N-Queens Problem (Backtracking)

Logic Flow:

1. Place queens row by row

2. For each row → try placing queen in column

3. Check if safe (no queen attacks in column/diagonal)

4. If safe → place queen, recurse for next row

5. If all rows done → solution found

6. Else → backtrack

Pseudo Code:

```
FUNCTION solveNQueens(board, row)
    IF row == N THEN PRINT board; RETURN True
    FOR col = 0 TO N-1
        IF safe(board, row, col) THEN
            board[row][col] = 1
            IF solveNQueens(board, row+1) THEN RETURN True
            board[row][col] = 0  // backtrack
    RETURN False
END
```

30□Trapping Rainwater Problem

Logic Flow:

1. Input array height[n]

2. Compute left_max[i] → max height to left of i

3. Compute right_max[i] → max height to right of i

4. Water trapped at i → min(left_max[i], right_max[i]) - height[i]

5. Sum for all i → total water

Pseudo Code:

```
START
Input height[n]
FOR i = 0 TO n-1: left_max[i] = max(height[0..i])
FOR i = n-1 TO 0: right_max[i] = max(height[i..n-1])
water = 0
FOR i = 0 TO n-1: water += min(left_max[i], right_max[i]) - height[i]
PRINT water
END
```

Note:
Advanced programs mostly use trees, stacks, heaps, recursion, backtracking, dynamic programming, sliding window techniques.