Slot **1** – Question **2 [Towers of Hanoi variant]** Marks 20

Gist: Classic TOH problem with restricted move – you can't move a disc from A to C and vice-versa.

Tasks:

- Print the correct recursion for TOH.
- In case an A to C or reverse move occurs in the original recursion, do that step using some more steps of recursions.

Grading Scheme: MANUAL

ANY FORM OF HARD-CODING ATTRACTS FULL PENALTY.

[3 Marks] : Correct recursion idea of classic TOH – i.e., the 3 steps are correct

[5 Marks]: Handling the special case of a direct move from A to C or reverse. It needs to be broken up into another set of three steps of recursive calls using B as auxiliary.

[2 Marks] : Correct termination condition.

======= GOLD CODE ================

```
#include<stdio.h>
// Hanoi Variant:
//
// Solve the original Towers of Hanoi problem, but with the
// extra restriction that you are not allowed to directly
// transfer a disk from A to C or from C to A. In other
// words, each movement must invlove B.

// the basic move of one disk: DO NOT CHANGE THIS FUNCTION.
void move(char From, char To) {
    const int NumPerLine = 8;
    static int num = 0;
    if (num%NumPerLine == 0) {
        printf("\n%5d: ", num);
    }
    else {
        printf(" ");
    }
    printf("%c->%c", From, To);
    num++;
}

// move n disks From A to C using B as auxx
// No direct A->C or C->A moves allowed
const char realA = 'A';
const char realC = 'C';
```

```
int hanoi(int n, char A, char C, char B) {
    int count = 0;
    if (n==0) { return 0; } // nothing to move!!

    // recursively move n-1 disks
    if (((A == realA) && (C == realC))
        ||((A == realC) && (C == realA))) {
        //case 1: if movement is from realA to realC or reverse:

        // from A to C using B as auxx
        count += hanoi(n-1, A, C, B);
        // move of LAST disk, through realB
        move(A, B);
        count += hanoi(n-1, C, A, B);
        move(B, C);
        count += 2;
        // from A to C using B as auxx
        count += hanoi(n-1, A, C, B);
    }
    else {
        //case 2: if movement is between other poles

        // from A to B using C as auxx
        count += hanoi(n-1, A, B, C);
        // move of LAST disk
        move(A, C);
        count++;
        // from B to C using A as auxx
        count += hanoi(n-1, B, C, A);
    }

    return count;
}

int main() {
    int n;
    scanf("%d", &n);
    hanoi(n, 'A', 'C', 'B');

    return 0;
}
```