1. Find the n-th Catalan Number using recursion

Problem: The Catalan number sequence is used in various combinatorial problems like counting BSTs, balanced parentheses, etc.
Formula:

Cn=∑i=0n−1Ci⋅Cn−i−1C_n = \sum_{i=0}^{n-1} C_i \cdot C_{n-i-1}Cn=i=0∑n−1Ci⋅Cn−i−1

Input-Output Format

makefile
CopyEdit

```
Input: 4
Output: 14
```

Solution

c
CopyEdit

```c
#include <stdio.h>

int catalan(int n) {
    if (n <= 1) return 1;
    int sum = 0;
    for (int i = 0; i < n; i++) {
        sum += catalan(i) * catalan(n - i - 1);
    }
    return sum;
}

int main() {
    int n;
    scanf("%d", &n);
    printf("%d\n", catalan(n));
    return 0;
}
```

---

2. Count ways to climb n stairs using recursion

Problem: You can take **1-step or 2-steps** at a time. Count the ways to reach the top.

Input-Output Format
makefile
CopyEdit
```
Input: 4
Output: 5
```

Solution
c
CopyEdit
```c
#include <stdio.h>

int countWays(int n) {
    if (n == 0) return 1;
    if (n < 0) return 0;
    return countWays(n - 1) + countWays(n - 2);
}

int main() {
    int n;
    scanf("%d", &n);
    printf("%d\n", countWays(n));
    return 0;
}
```

---

3. Sum of product of all subsets

Problem: Find the sum of the product of all subsets of an array.

Input-Output Format
makefile
CopyEdit
```
Input: 3
```

```
       1 2 3
Output: 23
```

Solution

c
CopyEdit

```
#include <stdio.h>

int subsetProductSum(int arr[], int n, int index, int product) {
    if (index == n) return product;
    return subsetProductSum(arr, n, index + 1, product) +
           subsetProductSum(arr, n, index + 1, product *
arr[index]);
}

int main() {
    int n, arr[20];
    scanf("%d", &n);
    for (int i = 0; i < n; i++) scanf("%d", &arr[i]);
    printf("%d\n", subsetProductSum(arr, n, 0, 1) - 1); //
Excluding empty subset
    return 0;
}
```

---

4. Find the longest common subsequence (LCS)

Problem: Given two strings, find the length of the **longest common subsequence**.

Input-Output Format

makefile
CopyEdit

```
Input: ABCD AEBD
Output: 3
```

Solution

c

CopyEdit

```
#include <stdio.h>
#include <string.h>

int lcs(char *X, char *Y, int m, int n) {
    if (m == 0 || n == 0) return 0;
    if (X[m - 1] == Y[n - 1])
        return 1 + lcs(X, Y, m - 1, n - 1);
    else
        return (lcs(X, Y, m - 1, n) > lcs(X, Y, m, n - 1)) ?
lcs(X, Y, m - 1, n) : lcs(X, Y, m, n - 1);
}

int main() {
    char X[100], Y[100];
    scanf("%s %s", X, Y);
    printf("%d\n", lcs(X, Y, strlen(X), strlen(Y)));
    return 0;
}
```

---

5. Find the nth Super Fibonacci Number

Problem: A **Super Fibonacci** sequence where each term is the sum of the previous **three terms**.

Input-Output Format

makefile

CopyEdit

```
Input: 6
Output: 24
```

Solution

c

CopyEdit

```
#include <stdio.h>

int superFibonacci(int n) {
    if (n == 1 || n == 2) return 1;
    if (n == 3) return 2;
    return superFibonacci(n - 1) + superFibonacci(n - 2) +
superFibonacci(n - 3);
}

int main() {
    int n;
    scanf("%d", &n);
    printf("%d\n", superFibonacci(n));
    return 0;
}
```

Advanced Problems (11-20)

---

6. Solve Tower of Hanoi problem

Problem: Move **n disks** from **source** to **destination** using an auxiliary peg.

Input-Output Format
vbnet
CopyEdit

```
Input: 3
Output:
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
```

```
Move disk 1 from A to C
```

Solution

c
CopyEdit

```c
#include <stdio.h>

void hanoi(int n, char src, char aux, char dest) {
    if (n == 1) {
        printf("Move disk 1 from %c to %c\n", src, dest);
        return;
    }
    hanoi(n - 1, src, dest, aux);
    printf("Move disk %d from %c to %c\n", n, src, dest);
    hanoi(n - 1, aux, src, dest);
}

int main() {
    int n;
    scanf("%d", &n);
    hanoi(n, 'A', 'B', 'C');
    return 0;
}
```

---

7. Generate all possible permutations of a string

Problem: Print all possible **permutations** of a given string.

Input-Output Format

makefile
CopyEdit

```
Input: ABC
Output:
ABC
ACB
```

```
BAC
BCA
CAB
CBA
```

Solution

c
CopyEdit

```c
#include <stdio.h>
#include <string.h>

void swap(char *x, char *y) {
    char temp = *x;
    *x = *y;
    *y = temp;
}

void permute(char *str, int l, int r) {
    if (l == r) {
        printf("%s\n", str);
        return;
    }
    for (int i = l; i <= r; i++) {
        swap(&str[l], &str[i]);
        permute(str, l + 1, r);
        swap(&str[l], &str[i]);
    }
}

int main() {
    char str[10];
    scanf("%s", str);
    permute(str, 0, strlen(str) - 1);
    return 0;
}
```

8. Find all subsets of a given set

Problem: Given a set of distinct integers, print all its subsets.

Input-Output Format

css
CopyEdit

```
Input: 3
       1 2 3
Output:
{}
{1}
{2}
{3}
{1, 2}
{1, 3}
{2, 3}
{1, 2, 3}
```

Solution

c
CopyEdit

```c
#include <stdio.h>

void printSubset(int arr[], int subset[], int n, int index, int
subsetSize) {
    printf("{");
    for (int i = 0; i < subsetSize; i++) {
        printf("%d", subset[i]);
        if (i < subsetSize - 1) printf(", ");
    }
    printf("}\n");

    for (int i = index; i < n; i++) {
        subset[subsetSize] = arr[i];
        printSubset(arr, subset, n, i + 1, subsetSize + 1);
```

```
    }
}

int main() {
    int n, arr[10], subset[10];
    scanf("%d", &n);
    for (int i = 0; i < n; i++) scanf("%d", &arr[i]);
    printSubset(arr, subset, n, 0, 0);
    return 0;
}
```

---

9. Print all combinations of balanced parentheses

Problem: Given n, print all valid **n pairs** of balanced parentheses.

Input-Output Format

bash
CopyEdit

```
Input: 3
Output:
((()))
(()())
(())()
()(())
()()()
```

Solution

c
CopyEdit

```
#include <stdio.h>

void generateParenthesis(int n, int open, int close, char str[],
int index) {
    if (close == n) {
        str[index] = '\0';
```

```
        printf("%s\n", str);
        return;
    }
    if (open < n) {
        str[index] = '(';
        generateParenthesis(n, open + 1, close, str, index + 1);
    }
    if (close < open) {
        str[index] = ')';
        generateParenthesis(n, open, close + 1, str, index + 1);
    }
}

int main() {
    int n;
    scanf("%d", &n);
    char str[2 * n + 1];
    generateParenthesis(n, 0, 0, str, 0);
    return 0;
}
```

---

10. Find the Josephus problem solution

Problem: n people stand in a circle. Every k-th person is eliminated until one remains.

Input-Output Format

makefile
CopyEdit

```
Input: 7 3
Output: 3
```

Solution

c
CopyEdit

```
#include <stdio.h>
```

```
int josephus(int n, int k) {
    if (n == 1) return 0;
    return (josephus(n - 1, k) + k) % n;
}

int main() {
    int n, k;
    scanf("%d %d", &n, &k);
    printf("%d\n", josephus(n, k));
    return 0;
}
```

---

11. Find the nth Ugly Number

Problem: An **ugly number** is only divisible by `2, 3, 5`. Find the `n-th` ugly number.

Input-Output Format

makefile
CopyEdit
`Input: 10`
`Output: 12`

Solution

c
CopyEdit

```
#include <stdio.h>

int min(int a, int b, int c) {
    return (a < b) ? (a < c ? a : c) : (b < c ? b : c);
}

int uglyNumber(int n) {
    int ugly[n], i2 = 0, i3 = 0, i5 = 0;
    int next2 = 2, next3 = 3, next5 = 5, nextUgly = 1;
```

```
    ugly[0] = 1;
    for (int i = 1; i < n; i++) {
        nextUgly = min(next2, next3, next5);
        ugly[i] = nextUgly;
        if (nextUgly == next2) next2 = ugly[++i2] * 2;
        if (nextUgly == next3) next3 = ugly[++i3] * 3;
        if (nextUgly == next5) next5 = ugly[++i5] * 5;
    }
    return ugly[n - 1];
}

int main() {
    int n;
    scanf("%d", &n);
    printf("%d\n", uglyNumber(n));
    return 0;
}
```

---

12. Reverse a linked list using recursion

Problem: Given a **linked list**, reverse it using recursion.

Input-Output Format

rust
CopyEdit

```
Input: 1 -> 2 -> 3 -> 4 -> NULL
Output: 4 -> 3 -> 2 -> 1 -> NULL
```

Solution

c
CopyEdit

```
#include <stdio.h>
#include <stdlib.h>
```

```
struct Node {
    int data;
    struct Node* next;
};

void printList(struct Node* head) {
    while (head) {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

struct Node* reverseList(struct Node* head) {
    if (!head || !head->next) return head;
    struct Node* newHead = reverseList(head->next);
    head->next->next = head;
    head->next = NULL;
    return newHead;
}

void push(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
    newNode->data = data;
    newNode->next = *head;
    *head = newNode;
}

int main() {
    struct Node* head = NULL;
    push(&head, 4);
    push(&head, 3);
    push(&head, 2);
    push(&head, 1);
```

```
    printList(head);
    head = reverseList(head);
    printList(head);

    return 0;
}
```

---

13. Find the median of two sorted arrays

Problem: Given **two sorted arrays**, find their **median**.

Input-Output Format

makefile
CopyEdit

```
Input: [1, 3, 5] [2, 4, 6]
Output: 3.5
```

Solution

c
CopyEdit

```
#include <stdio.h>

float median(int arr1[], int arr2[], int n) {
    int i = 0, j = 0, count, m1 = -1, m2 = -1;
    for (count = 0; count <= n; count++) {
        if (i != n && (j == n || arr1[i] <= arr2[j])) {
            m1 = m2;
            m2 = arr1[i++];
        } else {
            m1 = m2;
            m2 = arr2[j++];
        }
    }
    return (m1 + m2) / 2.0;
```

```
}

int main() {
    int arr1[3] = {1, 3, 5};
    int arr2[3] = {2, 4, 6};
    printf("%.1f\n", median(arr1, arr2, 3));
    return 0;
}
```

13. Find the median of two sorted arrays

Problem: Given **two sorted arrays**, find their **median**.

Input-Output Format

makefile
CopyEdit

```
Input:
3
1 3 5
3
2 4 6

Output: 3.5
```

Solution

c
CopyEdit

```
#include <stdio.h>

float median(int arr1[], int arr2[], int n) {
    int i = 0, j = 0, count, m1 = -1, m2 = -1;
    for (count = 0; count <= n; count++) {
        if (i != n && (j == n || arr1[i] <= arr2[j])) {
            m1 = m2;
            m2 = arr1[i++];
        } else {
```

```
            m1 = m2;
            m2 = arr2[j++];
        }
    }
    return (m1 + m2) / 2.0;
}

int main() {
    int n, arr1[100], arr2[100];
    scanf("%d", &n);
    for (int i = 0; i < n; i++) scanf("%d", &arr1[i]);
    for (int i = 0; i < n; i++) scanf("%d", &arr2[i]);

    printf("%.1f\n", median(arr1, arr2, n));
    return 0;
}
```

---

14. Compute binomial coefficient (nCr) using recursion

Problem: Compute **nCr**, which represents the number of ways to choose `r` elements from n.

Input-Output Format

makefile
CopyEdit
`Input: 5 2`
`Output: 10`

Solution

c
CopyEdit

```
#include <stdio.h>

int binomialCoeff(int n, int r) {
```

```
    if (r == 0 || r == n) return 1;
    return binomialCoeff(n - 1, r - 1) + binomialCoeff(n - 1,
r);
}

int main() {
    int n, r;
    scanf("%d %d", &n, &r);
    printf("%d\n", binomialCoeff(n, r));
    return 0;
}
```

---

15. Find the sum of digits of a number using recursion

Problem: Given a number, find the sum of its digits recursively.

Input-Output Format

makefile
CopyEdit
```
Input: 12345
Output: 15
```

Solution

c
CopyEdit
```
#include <stdio.h>

int sumOfDigits(int n) {
    if (n == 0) return 0;
    return (n % 10) + sumOfDigits(n / 10);
}

int main() {
    int n;
```

```
    scanf("%d", &n);
    printf("%d\n", sumOfDigits(n));
    return 0;
}
```

---

16. Generate all permutations of a string using recursion

Problem: Given a string, print all its permutations.

Input-Output Format

makefile
CopyEdit
```
Input: ABC
Output:
ABC
ACB
BAC
BCA
CAB
CBA
```

Solution

c
CopyEdit
```
#include <stdio.h>
#include <string.h>

void swap(char *x, char *y) {
    char temp = *x;
    *x = *y;
    *y = temp;
}

void permute(char *str, int l, int r) {
```

```
    if (l == r) {
        printf("%s\n", str);
        return;
    }
    for (int i = l; i <= r; i++) {
        swap((str + l), (str + i));
        permute(str, l + 1, r);
        swap((str + l), (str + i)); // Backtrack
    }
}

int main() {
    char str[10];
    scanf("%s", str);
    permute(str, 0, strlen(str) - 1);
    return 0;
}
```

---

17. Find the power of a number using recursion

Problem: Compute a^b using recursion.

Input-Output Format

makefile
CopyEdit

```
Input: 2 5
Output: 32
```

Solution

c
CopyEdit

```
#include <stdio.h>

int power(int a, int b) {
```

```
    if (b == 0) return 1;
    return a * power(a, b - 1);
}

int main() {
    int a, b;
    scanf("%d %d", &a, &b);
    printf("%d\n", power(a, b));
    return 0;
}
```

---

18. Reverse an array using recursion

Problem: Reverse an array recursively.

Input-Output Format

makefile
CopyEdit

```
Input:
5
1 2 3 4 5

Output:
5 4 3 2 1
```

Solution

c
CopyEdit

```
#include <stdio.h>

void reverseArray(int arr[], int start, int end) {
    if (start >= end) return;
    int temp = arr[start];
    arr[start] = arr[end];
```

```
    arr[end] = temp;
    reverseArray(arr, start + 1, end - 1);
}

int main() {
    int n, arr[100];
    scanf("%d", &n);
    for (int i = 0; i < n; i++) scanf("%d", &arr[i]);

    reverseArray(arr, 0, n - 1);

    for (int i = 0; i < n; i++) printf("%d ", arr[i]);
    printf("\n");
    return 0;
}
```

---

19. Solve Tower of Hanoi problem

Problem: Move n disks from A to C using B as an auxiliary rod.

Input-Output Format
vbnet
CopyEdit
```
Input: 3
Output:
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
```

Solution

c
CopyEdit

```c
#include <stdio.h>

void towerOfHanoi(int n, char from, char to, char aux) {
    if (n == 1) {
        printf("Move disk 1 from %c to %c\n", from, to);
        return;
    }
    towerOfHanoi(n - 1, from, aux, to);
    printf("Move disk %d from %c to %c\n", n, from, to);
    towerOfHanoi(n - 1, aux, to, from);
}

int main() {
    int n;
    scanf("%d", &n);
    towerOfHanoi(n, 'A', 'C', 'B');
    return 0;
}
```

---

20. Count ways to reach the nth stair

Problem: Given n stairs, find the number of ways to climb them, taking 1 or 2 steps at a time.

Input-Output Format

makefile
CopyEdit

```
Input: 4
Output: 5
```

Solution

c
CopyEdit

```c
#include <stdio.h>

int countWays(int n) {
    if (n == 1) return 1;
    if (n == 2) return 2;
    return countWays(n - 1) + countWays(n - 2);
}

int main() {
    int n;
    scanf("%d", &n);
    printf("%d\n", countWays(n));
    return 0;
}
```