PROBLEM SOLVING TRICKS

C Operators Tricks:

Category	Trick	Description
Bitwise	if (num & 1)	Checks if a number is odd (returns 1 if odd, 0 if even).
	num & (divisor - 1)	Fast modulo operation when divisor is a power of 2.
	if (n > 0 && (n & (n - 1)) == 0)	Check if a number is a power of 2
	num = num | (1 << k);	Set a specific bit in a number.
	a ^= b; b ^= a; a ^= b;	Swaps two variables without a temporary variable using XOR.
	num << 1	Multiplies a number by 2 (left shift).
	num >> 1	Divides a number by 2 (right shift).
	while (n) { count += n & 1; n >>= 1; }	Count set bits in an integer.
	(x ^ y) < 0	Checks if two numbers have opposite signs.
Bitwise & Arithmetic	(num ^ (num >> 31)) - (num >> 31)	Computes absolute value without branching.
	y ^ ((x ^ y) & -(x < y))	Finds the minimum of two numbers without if-else.
	x ^ ((x ^ y) & -(x < y))	Finds the maximum of two numbers without if-else.
Logical	max = (a > b) ? a : b;	Find the maximum of two numbers.
	!!boolean_value	Converts a boolean to 1 (true) or 0 (false).
Pointers	*(arr + i) = value	Alternative way to access array elements using pointer arithmetic.
	ptr2 - ptr1	Calculates the number of elements between two pointers.

Miscellaneous	1 << n	Computes 2n2n (fast power of 2).
	num && !(num & (num - 1))	Checks if a number is a power of 2.
	ch ^= 32	Toggles the case of an alphabet character (uppercase ↔ lowercase).
	Brian Kernighan's Algorithm (num &= (num - 1))	Counts the number of set bits (1s) in an integer efficiently.
	a = (b, b = a, a);	Swap and assign in one line.

C Operators Problem-Solving Questions:

SNO	PROBLEM	SOLUTION
1	Check if a number is even or odd	Use & if (num & 1) printf("Odd"); else printf("Even");
2	Toggle the k-th bit of a number	Use ^ num = num ^ (1 << k);
3	Count trailing zeros in binary	Use >> and & while ((n & 1) == 0) { count++; n >>= 1; }

4	Fast multiplication/division by 2	Use << and >>
		int mul = x << 3; // x * 8
		int div = x >> 2; // x / 4

5	Check if a number is a power of 2	if (n != 0 && (n & (n - 1)) == 0) printf("Power of 2");
6	Check if a number is a power of 4	Use & and % if ((n & (n - 1)) == 0 && n % 3 == 1) { /* Yes */ }
7	Toggle a bit at position k	n = n ^ (1 << k);
8	Set a bit at position k	n = n | (1 << k);
9	Clear a bit at position k	n = n & ~(1 << k);

10	Modulo of power of 2 (faster)	int mod = x & (n - 1); // only when n is power of 2
		Faster than x % n

11	Quick sign detection	if ((x >> 31) & 1)
		  printf("Negative");
		else
		  printf("Positive or Zero");

12	Swap case of a character using bitwise	ch = ch ^ 32; Works if ch is an ASCII alphabet letter

13	Check if two numbers have opposite signs	if ((a ^ b) < 0)
		  printf("Opposite signs");
		XOR of numbers with opposite signs is negative
		No need to check a < 0 && b > 0 || a > 0 && b < 0

14	Get the rightmost set bit	int r = n & (-n); Useful for bitmask-based dynamic programming Fast way to isolate least significant 1
15	Turn off the rightmost set bit	n = n & (n - 1); Used in Brian Kernighan's algorithm Useful in subset generation
16	Find the sign of an integer	int sign = (x >> 31) | (!!x); sign = -1 for negative, 1 for positive, 0 for zero Clever and compact
17	Convert uppercase to lowercase using bitwise	ch = ch | 32; // A-Z to a-z

18	Convert lowercase to uppercase	ch = ch & ~32; // a-z to A-Z
		Works only on ASCII letters
		Bit-level case conversion

19	Check if a number is positive without using >	if (!(x >> 31) && x != 0) printf("Positive");

20	Check if a number is divisible by 9	while (n > 9)
		  n = n / 10 + n % 10;
		if (n == 9)
		  printf("Divisible by 9");

		Digit sum trick Can be optimized using recursion or loop

21	Count set bits (Brian Kernighan's Algorithm)	int count = 0;
		while (n) {
		  n = n & (n - 1);
		  count++;
		}
		Very efficient for counting 1's in binary

C Conditional Statements: Tricks for Problem Solving

1. Simple if Statement

- Use Case: Execute code only if a condition is true.

Tricks:

- Avoid redundant checks – Exit early if possible.
- Use logical operators (&&, ||) to combine conditions efficiently.

Ex: Check if a number is positive	Optimization Trick

if (num > 0) {
  printf("Positive");
}
	if (num <= 0) return; // Early exit for invalid cases
	// Rest of the logic for positive numbers

2. if-else Statement

- Use Case: Choose between two alternatives.

Tricks:

- Ternary operator (?:) for simple assignments.
- Order conditions wisely – Place the most likely condition first.

Ex: Even or Odd Check	Optimization Trick

if (num % 2 == 0)
  printf("Even");
else
  printf("Odd");
	// Ternary Operator
	printf(num % 2 ? "Odd" : "Even");

3. else-if Ladder

- ❖ Use Case: Multiple exclusive conditions.
- ❖ Order conditions from most to least probable for efficiency.
- ❖ Use switch if comparing a single variable against constants.

Ex: Grade Classification	Optimization Trick

if (marks >= 90) {
  printf("A");
} else if (marks >= 80) {
  printf("B");
} else if (marks >= 70) {
  printf("C");
} else {
  printf("Fail");
}
	// Early return
	if (marks >= 90) return "A";
	if (marks >= 80) return "B";
	if (marks >= 70) return "C";
	return "Fail";

4. Nested if Statements

Use Case: Hierarchical conditions (one condition inside another).

Tricks:

- ❖ Flatten nested ifs when possible for readability.
- ❖ Use logical operators (&&, ||) to combine conditions.

Example: Check if a number is in a range and even

```
if (num >= 10 && num <= 100) {
  if (num % 2 == 0) {
```

```
        printf("Valid even number in range");
    }
}
```

Optimization Trick (Combined Conditions)

```
if (num >= 10 && num <= 100 && num % 2 == 0) {
    printf("Valid even number in range");
}
```

Switch Statement Tricks

Use Case: Compare a variable against multiple constant values.

Tricks:

- ❖ Use switch instead of long else-if ladders for better readability.
- ❖ Always include default case for unexpected values.
- ❖ Use fallthrough (no break) for multiple cases with the same logic.

Example: Day of the Week

```
switch (day) {
    case 1: printf("Monday"); break;
    case 2: printf("Tuesday"); break;
    case 3: printf("Wednesday"); break;
    default: printf("Invalid day");
}
```

Optimization Trick (Fallthrough Cases)

```
switch (month) {
    case 1: case 3: case 5: case 7: case 8: case 10: case 12:
        printf("31 days"); break;
    case 4: case 6: case 9: case 11:
        printf("30 days"); break;
    case 2:
        printf("28/29 days"); break;
    default: printf("Invalid month");}
```

Comparison Table: When to Use Which?

Scenario	Best Statement	Example
Single condition check	If	if (temperature > 30)
Two possible outcomes	if-else	Even/odd check
Multiple mutually exclusive conditions	else-if ladder	Grade assignment
Conditions dependent on other conditions	Nested if	"Valid positive even number"
Comparing a variable against many constants	switch	Day/month names

Switch with Enums for Readability

Trick: Use enum with switch for cleaner code.

Example: Handling Command Inputs

```
enum Command { START, STOP, PAUSE, QUIT };
enum Command cmd = get_command();
switch (cmd) {
    case START: start(); break;
    case STOP: stop(); break;
    case PAUSE: pause(); break;
    case QUIT: exit(0);
    default: printf("Unknown command");}
```

1. Avoid Numbers use Enum or #define

AVOID	USE

If(gender==1)
Printf(“male code”)
If(gender==0)
Printf(“female code”)
	#define MALE 1
	#define FEMALE 0
	If(gender==MALE)
	Printf(“male code”)
	OR
	enum gender{FEMALE,MALE}

2. Use switch-case for Multiple Conditions

AVOID	USE

if (day == 1) printf("Monday");
else if (day == 2)
printf("Tuesday");
else if (day == 3)
printf("Wednesday");
// ... up to 7
	switch (day) {
	case 1: printf("Monday");
	break;
	case 2: printf("Tuesday");
	break;
	case 3: printf("Wednesday");
	break;
	// ...
	default: printf("Invalid day");
	}

3. Use Lookup Tables Instead of Long if-else

AVOID	USE

if (month == 1) days = 31;
else if (month == 2) days = 28;
else if (month == 3) days = 31;
// ... up to 12
	int days_in_month[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
	days = days_in_month[month - 1];

4. Early Exit for Performance Optimization

AVOID	USE

for (int i = 0; i < n; i++) {
if (arr[i] == target) {
found = true;
}
}
	for (int i = 0; i < n; i++) {
	if (arr[i] == target) {
	found = true;
	break; // Exit early
	}}

5. Null Checks

AVOID	USE

if (ptr->data == 42)
{
// Do something
}
	if(ptr!=NULL&&ptr->data==42)
	{
	// Do something
	}

6. Lazy Evaluation for Expensive Checks

AVOID	USE

if (is_valid(input) &&
heavy_computation(input))
{
// Do something
}
	if (is_valid(input))
	{
	if (heavy_computation(input))
	{
	// Only compute if needed
	// Do something
	}}

7. Loop Fusion (Combine Loops)

AVOID	USE

// Before (2 passes)
for (int i = 0; i < n; i++)
{
a[i] = b[i] + 1;
}
for (int i = 0; i < n; i++)
{
c[i] = a[i] * 2;}
	// After (1 pass)
	for (int i = 0; i < n; i++)
	{
	a[i] = b[i] + 1;
	c[i] = a[i] * 2;
	}

8. Loop-and-a-Half Pattern

When: Need to check condition mid-iteration

How: Move termination check inside

AVOID	USE

// Standard approach
while (true) {
data = get_input();
if (data == TERMINATOR) break;
process(data);
}
	// Alternative
	while ((data = get_input()) != TERMINATOR) {
	process(data);
	}

9. Loop Variable Reversal

When: Backward processing is needed

How: Count down instead of up

Why: Comparison with 0 is faster on many CPUs.

```
// More efficient for some cases
for (int i = n-1; i >= 0; i--)
{
    process(i);
}
```

BASIC PROBLEM SOLVING QUESTIONS

1	**PROBLEM**	Nim Game
	LINK	https://leetcode.com/problems/nim-game
2	**PROBLEM**	Bulb Switcher
	LINK	https://leetcode.com/problems/bulb-switcher
3	**PROBLEM**	Minimum Operations to Make the Integer Zero
	LINK	https://leetcode.com/problems/minimum-operations-to-make-the-integer-zero
4	**PROBLEM**	Dice Number
	LINK	https://www.codechef.com/problems/DICENUM
5	**PROBLEM**	Largest K
	LINK	https://www.codechef.com/problems/LARGESTK343
6	**PROBLEM**	Subset Sum 3
	LINK	https://www.codechef.com/problems/SUBSUM3
7	**PROBLEM**	Two Ranges
	LINK	https://www.codechef.com/problems/TWORANGES
8	**PROBLEM**	Power(x,y)
	LINK	https://leetcode.com/problems/powx-n
9	**PROBLEM**	GCD(x,y)
	LINK	https://leetcode.com/problems/find-greatest-common-divisor-of-array

10	**PROBLEM**	Number of 1 Bits
	LINK	https://leetcode.com/problems/number-of-1-bits
11	**PROBLEM**	Diamond pattern
	LINK	https://takeuforward.org/pattern/pattern-9-diamond-star-pattern
12	**PROBLEM**	Pascal triangle
	LINK	https://leetcode.com/problems/pascals-triangle
13	**PROBLEM**	Inverted star pyramid pattern
	LINK	https://takeuforward.org/pattern/pattern-8-inverted-star-pyramid
14	**PROBLEM**	Number pyramid
	LINK	https://takeuforward.org/pattern/pattern-3-right-angled-number-pyramid
15	**PROBLEM**	Binary palindrome
	LINK	https://www.geeksforgeeks.org/check-binary-representation-number-palindrome
16	**PROBLEM**	Happy number
	LINK	https://leetcode.com/problems/happy-number
17	**PROBLEM**	Prime Factorization
	LINK	-
18	**PROBLEM**	Perfect Squares Between Range
	LINK	https://www.geeksforgeeks.org/dsa/find-number-perfect-squares-two-given-numbers

19	**PROBLEM**	Josephus Problem
	LINK	https://takeuforward.org/data-structure/josephus-problem
20	**PROBLEM**	Frequency Count without Array
	LINK	-
21	**PROBLEM**	Add two binary numbers using loops only (no bitwise or built-in functions)
	LINK	https://www.geeksforgeeks.org/java-program-to-add-two-binary-strings
22	**PROBLEM**	Print time from 00:00 to 23:59 using nested loops
	LINK	-
23	**PROBLEM**	Print monthly calendar using nested loops and date logic
	LINK	https://www.geeksforgeeks.org/python-program-to-display-calendar-using-loop
24	**PROBLEM**	Smallest number with at least N trailing zeroes in factorial
	LINK	https://www.geeksforgeeks.org/smallest-number-least-n-trailing-zeroes-factorial
25	**PROBLEM**	Reverse words in a string
	LINK	https://leetcode.com/problems/reverse-words-in-a-string
26		