

A competitive programmer’s Handbook



Welcome to the ultimate journey into the world of Competitive Programming and Data Structures. Whether you're a beginner or looking to sharpen your skills, this course is designed to equip you with the tools, techniques, and mindset to solve complex problems efficiently.

Let's dive into logic, algorithms, and beyond!

**Bit manipulation**

**Binary number**

A **binary number** is a number expressed in the base-2 numeral system or binary numeral system, it is a method of mathematical expression which uses only two symbols: typically "0" (zero) and "1" (one).

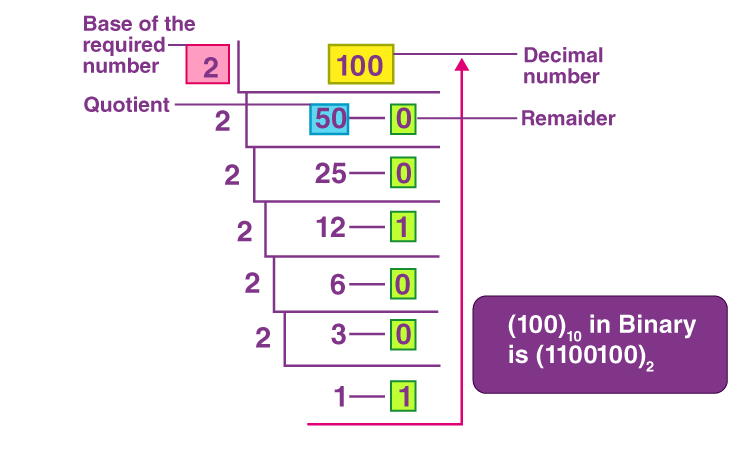
**NOTE:**

Conversion of number from one base to other can be done as:

Base x 🡪 Base 10 🡪 Base y

**Steps to Convert Decimal (100) to Binary:**

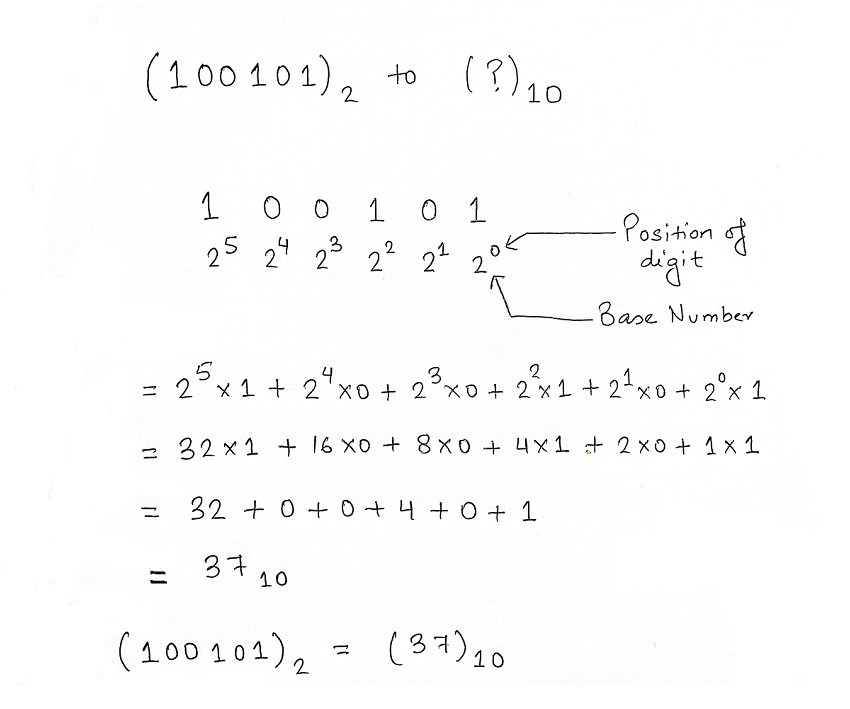
1. **Start with the decimal number.**  
   Example: **100**.
2. **Divide the number by 2.**  
   Write down the **remainder**:
   * 100 ÷ 2 = 50, remainder = **0**.
3. **Repeat the division with the quotient.**
   * 50 ÷ 2 = 25, remainder = **0**.
   * 25 ÷ 2 = 12, remainder = **1**.
   * 12 ÷ 2 = 6, remainder = **0**.
   * 6 ÷ 2 = 3, remainder = **0**.
   * 3 ÷ 2 = 1, remainder = **1**.
   * 1 ÷ 2 = 0, remainder = **1**.
4. **Write the remainders in reverse order.**  
   Remainders (from bottom to top): **1, 1, 0, 0, 1, 0, 0**.  
   Binary representation of 100 is **1100100**.



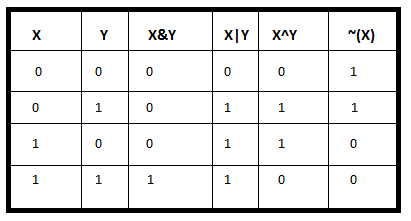
**Steps to Convert Binary (100101) to Decimal:**

1. **Write down the binary number.**  
   Example: **100101**.
2. **Assign powers of 2 to each digit.**  
   From right to left:
3. **Multiply each binary digit by its power of 2.**
4. **Add the results.**  
   32+0+0+4+0+1=3732 + 0 + 0 + 4 + 0 + 1 = 37.

**Binary 100101 = Decimal 37.**



**Bit operators:**



* **AND &**: Compares bits of two numbers. If **both bits are 1**, the result is **1**; otherwise, it's **0**.

n = 01011000

n-1 = 01010111

----------------------------

n & (n-1) = 01010000

* **OR**   **|**: Compares bits of two numbers. If **either bit is 1**, the result is **1**; otherwise, it's **0**.

n = 01011000

n-1 = 01010111

----------------------------------

n | (n-1) = 01011111

* **XOR** **∧**: Compares bits of two numbers. If the bits are **different**, the result is **1**; if they’re the **same**, the result is **0**.

n = 01011000

n-1 = 01010111

--------------------

n ^ (n-1) = 00001111

* **NOT**   **∼**: Flips each bit of a number:
  + - If the bit is **1**, it becomes **0**.
  + If the bit is **0**, it becomes **1**.

n = 01011000

----------------------------

~n = 10100111

**One’s Complement**

1. Convert the number to binary.
2. Flip all bits (0 → 1, 1 → 0).
3. The result is the **one’s complement** of the number.

Eg: 00000000000000000000000000001100 (12 in binary)

11111111111111111111111111110011 (-12 in one's complement)

**Two's Complement**

**Step 1: Convert the Number to Binary**

* If the number is positive, write its binary representation normally.
* If the number is negative, first write its positive counterpart in binary.

**Step 2: Take the Bitwise Complement (~)**

* Flip all the bits (change 0 to 1 and 1 to 0).

**Step 3: Add 1 to the Complemented Binary Number**

* Perform binary addition of 1 to the flipped bits.

Example:

00001100 (Binary for 12)

11110011 (Flip all bits)

11110011

+ 00000001

------------

11110100 (Final result, which is -12 in 8-bit two’s complement)

**Properties of Bitwise Operators**

● OR , AND , XOR are associative and commutative.

● A^0 = A

● A^A = 0

● If A^B = C , then A^C = B and B^C = A

● A&B <= MIN(A,B)

● A|B >= MAX(A,B)

● (A&1) is 1 if A→odd , else 0

● A & (A-1) is 0 if A is power of 2

**Some tricks related to Bitwise Operators**

● Parity of number of set bits in A ^ B

● A + B = (A ^ B) + 2 \* (A & B)

● A + B = (A | B) + (A & B)

● Swap two numbers without using swap function and temporary

Variable.

**Inbuilt function in Bit Manipulation**

Bit manipulation in C and C++ has several inbuilt functions to efficiently perform operations like counting set bits, finding the highest/lowest set bit, and checking parity. Here are some commonly used functions:

**1. \_\_builtin\_popcount(x)**

* Counts the number of set bits (1s) in an integer.
* **Example:**
* int x = 13; // Binary: 1101
* int count = \_\_builtin\_popcount(x); // Returns 3
* Variants:
  + \_\_builtin\_popcountl(x) → For long
  + \_\_builtin\_popcountll(x) → For long long

**2. \_\_builtin\_clz(x)**

* Counts the number of **leading zeros** in an integer.
* **Example:**
* int x = 16; // Binary: 00000000000000000000000000010000
* int leading\_zeros = \_\_builtin\_clz(x); // Returns 27 (for 32-bit int)
* Variants:
  + \_\_builtin\_clzl(x) → For long
  + \_\_builtin\_clzll(x) → For long long

**3. \_\_builtin\_ctz(x)**

* Counts the number of **trailing zeros** in an integer.
* **Example:**
* int x = 8; // Binary: 00000000000000000000000000001000
* int trailing\_zeros = \_\_builtin\_ctz(x); // Returns 3
* Variants:
  + \_\_builtin\_ctzl(x) → For long
  + \_\_builtin\_ctzll(x) → For long long

**4. \_\_builtin\_parity(x)**

* Returns 1 if the number of set bits is odd, otherwise 0.
* **Example:**
* int x = 7; // Binary: 0111
* int parity = \_\_builtin\_parity(x); // Returns 1 (odd number of 1s)
* Variants:
  + \_\_builtin\_parityl(x) → For long
  + \_\_builtin\_parityll(x) → For long long

**5. \_\_builtin\_ffs(x)**

* Returns the position (1-based) of the **first set bit** from the **right**.
* **Example:**
* int x = 18; // Binary: 10010
* int first\_set\_bit = \_\_builtin\_ffs(x); // Returns 2

**Basic Questions:**

**Q1: Check if the ith bit is set**

bool isBitSet(int n, int i) {

return (n & (1 << i)) != 0;

}

**Q2: Set the ith bit of a number**

int setBit(int n, int i) {

return n | (1 << i);

}

**Q3: Clear the ith bit of a number**

int clearBit(int n, int i) {

return n & ~(1 << i);

}

**Q4: Remove the last set bit of a number**

int removeLastSetBit(int n) {

return n & (n - 1);

}

**Q5: Find whether a number is even or odd**

bool isEven(int n) {

return (n & 1) == 0;

}

**Q6: Check if the number is a power of 2**

bool isPowerOf2(int n) {

return n > 0 && (n & (n - 1)) == 0;

}

**Q7: Check if a number is a power of 4, 8, 16**

bool isPowerOf(int n, int base) {

while (n > 1) {

if (n % base != 0) return false;

n /= base;

}

return n == 1;

}

**Q8: Toggle ith Bit of a number**

int toggleBit(int n, int i) {

return n ^ (1 << i);

}

**Q9: Count the number of set bits**

int countSetBits(int n) {

int count = 0;

while (n) {

count += n & 1;

n >>= 1;

}

return count;

}

**Q10: Find the two non-repeating elements in an array**

vector<int> findUniqueNumbers(vector<int>& nums) {

int XOR = 0;

for (int num : nums) XOR ^= num;

int setBit = XOR & -XOR;

int x = 0, y = 0;

for (int num : nums) {

if (num & setBit) x ^= num;

else y ^= num;

}

return {x, y};

}

**Q11: Convert case operations**

char toLower(char c) { return c | ' '; }

char toUpper(char c) { return c & '\_'; }

char invertCase(char c) { return c ^ ' '; }

**Q12: Find letter position in alphabet**

int letterPosition(char c) {

return c & 31;

}

**Q13: Find the odd occurring number**

int findOddOccurrence(vector<int>& nums) {

int res = 0;

for (int num : nums) res ^= num;

return res;

}

**Q14: Swap two numbers using XOR**

void swap(int &a, int &b) {

a ^= b;

b ^= a;

a ^= b;

}

**Q15: XOR range calculations**

int XORupto(int n) {

return (n % 4 == 0) ? n : (n % 4 == 1) ? 1 : (n % 4 == 2) ? n + 1 : 0;

}

int XORrange(int L, int R) {

return XORupto(R) ^ XORupto(L - 1);

}

**Q16: Even check and XOR of subsets (Always 0 if array has >1 elements)**

bool isEvenNumber(int n) { return (n & 1) == 0; }

int XORofSubsets(vector<int>& nums) { return nums.empty() ? 0 : 0; }

**Q17: Count bits to flip A to B**

int countFlips(int a, int b) {

return countSetBits(a ^ b);

}

**Q18: Find missing number in array**

int findMissingNumber(vector<int>& nums, int n) {

int XOR = 0;

for (int i = 1; i <= n; i++) XOR ^= i;

for (int num : nums) XOR ^= num;

return XOR;

}

**Q19: Print binary representation**

void printBinary(int n) {

for (int i = 31; i >= 0; i--) cout << ((n >> i) & 1);

cout << endl;

}

**Q20: Reverse bits of a number**

int reverseBits(int n) {

int res = 0;

for (int i = 0; i < 32; i++) {

res = (res << 1) | (n & 1);

n >>= 1;

}

return res;

}

**Q21: Swap ith and jth bits**

int swapBits(int n, int i, int j) {

int bit1 = (n >> i) & 1, bit2 = (n >> j) & 1;

if (bit1 != bit2) n ^= (1 << i) | (1 << j);

return n;

}

**Q22: Swap all even and odd bits**

int swapEvenOddBits(int n) {

return ((n & 0xAAAAAAAA) >> 1) | ((n & 0x55555555) << 1);

**Q23: Copy and toggle bits in a range**

int copySetBits(int a, int b, int l, int r) {

int mask = ((1 << (r - l + 1)) - 1) << (l - 1);

return a | (b & mask);

}

int toggleBits(int n, int l, int r) {

int mask = ((1 << (r - l + 1)) - 1) << (l - 1);

return n ^ mask;

}

**Q24: Divide two integers without using multiplication, division, or mod**

int divide(int dividend, int divisor) {

int sign = ((dividend < 0) ^ (divisor < 0)) ? -1 : 1;

long long dvd = abs(dividend), dvs = abs(divisor), res = 0;

while (dvd >= dvs) {

long long temp = dvs, multiple = 1;

while (dvd >= (temp << 1)) {

temp <<= 1;

multiple <<= 1;

}

dvd -= temp;

res += multiple;

}

return sign \* res;

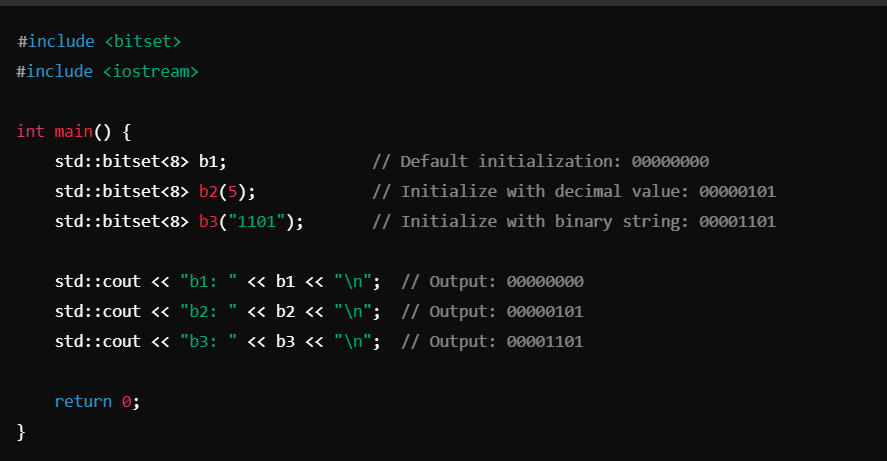
}

**Advanced concepts in BitManipulation**

**bitset in C++**

std::bitset is a container in C++ that allows you to store and manipulate a fixed-size sequence of bits efficiently. It provides an easy way to perform bitwise operations like setting, flipping, and counting bits.

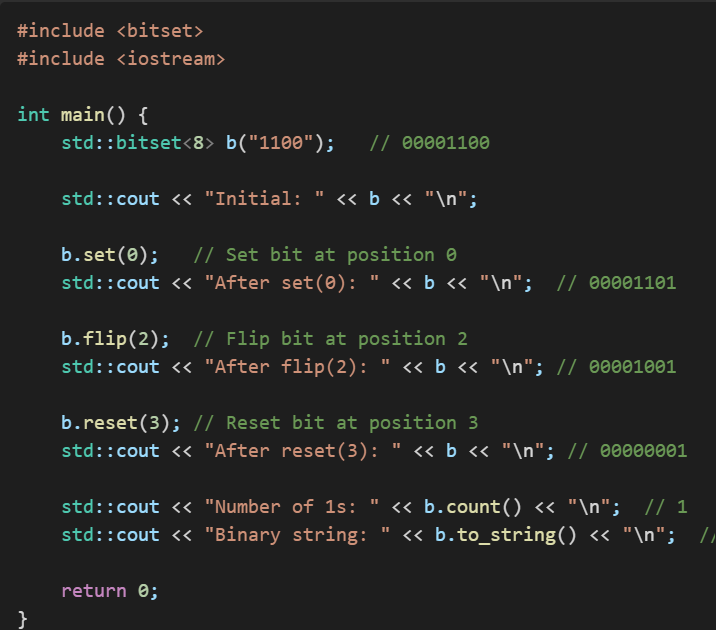
**1. Declaration and Initialization**



**2. Common Operations on bitset**

* **set(pos):** Sets bit at pos to 1. Example: b.set(2); (00000100 → 00000110)
* **set():** Sets all bits to 1. Example: b.set(); (00000000 → 11111111)
* **reset(pos):** Sets bit at pos to 0. Example: b.reset(2);
* **reset():** Sets all bits to 0. Example: b.reset();
* **flip(pos):** Toggles bit at pos. Example: b.flip(2);
* **flip():** Toggles all bits. Example: b.flip();
* **count():** Returns the number of set bits (1s). Example: b.count();
* **size():** Returns the total number of bits. Example: b.size();
* **test(pos):** Checks if bit at pos is 1. Example: b.test(2);
* **none():** Returns true if all bits are 0. Example: b.none();
* **any():** Returns true if at least one bit is 1. Example: b.any();
* **to\_ulong():** Converts bitset to unsigned long. Example: b.to\_ulong();
* **to\_string():** Converts bitset to a binary string. Example: b.to\_string();

1. **Example: Bit Manipulation Using bitset**

****

1. **Logical & Bitwise Operations**

****

**Dsa Questions:**

1)[Minimum Bit Flips to Convert Number](https://leetcode.com/problems/minimum-bit-flips-to-convert-number/)- <https://leetcode.com/problems/minimum-bit-flips-to-convert-number/description/> asked in **wells fargo**

2)First Set Bit

<https://www.geeksforgeeks.org/problems/find-first-set-bit-1587115620/1?itm_source=geeksforgeeks&itm_medium=article&itm_campaign=practice_card> asked in **PWC**

3) [Single Number](https://leetcode.com/problems/single-number/)

<https://leetcode.com/problems/single-number/description/>

4) [Subsets](https://leetcode.com/problems/subsets/)

<https://leetcode.com/problems/subsets/description/>

4) XORwice

<https://codeforces.com/problemset/problem/1421/A>

5) Raising Bacteria

<https://codeforces.com/problemset/problem/579/A>