This is one of the most important concepts in **Bitwise Operators**. The easiest way to understand it is by converting the numbers to **binary (bits)**.

Program

```
#include<stdio.h>

int main()
{
    printf("%d", 5 & 3);

    return 0;
}
```

Output

```
1
```

But **why is the answer 1?**

Step 1: Convert Decimal to Binary

First, write both numbers in binary.

```
Decimal     Binary

5           0101
3           0011
```

Let's align the bits:

```
      0 1 0 1   (5)

      0 0 1 1   (3)
```

Step 2: Understand the AND (&) Operator

The **Bitwise AND (&)** operator compares each bit.

Rule

```
0 & 0 = 0

0 & 1 = 0

1 & 0 = 0

1 & 1 = 1
```

Remember this simple sentence:

AND gives 1 only when BOTH bits are 1. Otherwise, it gives 0.

Step 3: Compare Each Bit

```
      0 1 0 1    (5)

&     0 0 1 1    (3)

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

      0 0 0 1
```

Let's compare bit by bit.

First Bit

```
0 & 0 = 0
```

Second Bit

```
1 & 0 = 0
```

Third Bit

```
0 & 1 = 0
```

Fourth Bit

```
1 & 1 = 1
```

Final Answer

```
0001
```

Step 4: Convert Binary Back to Decimal

```
Binary

0001
```

Calculate using powers of 2.

Bit Position	2^3	2^2	2^1	2^0
Binary	0	0	0	1
Value	0×8	0×4	0×2	1×1

```
= 0 + 0 + 0 + 1

= 1
```

Therefore,

```
5 & 3 = 1
```

Visual Explanation

```
Decimal

5 = 0101

3 = 0011

AND
------------
0101
0011
------------
0001

Decimal = 1
```

Real-Life Example (Easy to Remember)

Imagine two switches controlling a light.

```
Switch A      Switch B      Light

OFF              OFF         OFF

OFF              ON          OFF

ON               OFF         OFF

ON               ON          ON
```

The light turns **ON only when both switches are ON**.

This is exactly how the **AND (&)** operator works.

Another Example

Find:

```
6 & 3
```

Convert to binary.

```
6 = 0110

3 = 0011
0110

0011
------

0010
```

Binary `0010` equals:

```
2
```

So,

```
printf("%d", 6 & 3);
```

Output

```
2
```

Memory Trick for Students

```
Bitwise AND (&)

1 & 1 = 1 ✅

Everything else = 0 ❌
```

Or remember this phrase:

"AND means BOTH must be 1."

This single rule is enough to solve every Bitwise AND problem in C.

not clear Bitwise XOR (^) ?

This is a very common doubt among C programming students. The easiest way to understand **Bitwise XOR (^)** is **not by memorizing**, but by thinking of it as **"Different = 1, Same = 0."**

What is Bitwise XOR (^)?

XOR means **Exclusive OR**.

It compares **each bit** of two numbers.

Rule

```
0 ^ 0 = 0

0 ^ 1 = 1

1 ^ 0 = 1

1 ^ 1 = 0
```

Memory Trick

If the bits are the SAME → 0
If the bits are DIFFERENT → 1

Example 1

```
printf("%d",5^3);
```

Let's solve it step by step.

Step 1: Convert to Binary

```
Decimal      Binary

5            0101

3            0011
```

Step 2: Compare Each Bit

```
      0 1 0 1    (5)

^     0 0 1 1    (3)

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

      0 1 1 0
```

Let's compare one bit at a time.

5	3	XOR Result	Reason
0	0	0	Same
1	0	1	Different
0	1	1	Different
1	1	0	Same

Result

```
0110
```

Step 3: Convert Back to Decimal

```
0110
```

Bit	Value
0	8
1	4
1	2
0	1

```
4 + 2 = 6
```

Therefore,

```
5 ^ 3 = 6
```

Output

```
6
```

Easy Way to Remember

Imagine two students wearing shirts.

Student A

```
Red
```

Student B

```
Red
```

Same color

↓

```
0
```

Student A

```
Red
```

Student B

```
Blue
```

Different

↓

```
1
```

XOR always asks:

"Are they different?"

If Yes

```
1
```

If No

```
0
```

Another Example

Find

```
6 ^ 3
```

Binary

```
6 = 0110

3 = 0011
```

Compare

```
0110

0011

------

0101
```

Now convert

```
0101

=4+1

=5
```

Output : 5

One More Example

```
7 ^ 7
```

Binary

```
7 = 0111
```

```
7 = 0111
```

Compare

```
0111
```

```
0111
```

```
------
```

```
0000
```

Output

```
0
```

Because every bit is the **same**.

Compare AND, OR and XOR Together

Take the same numbers:

```
5 = 0101
```

```
3 = 0011
```

Bitwise AND (&)

Rule:

Both bits must be **1**

```
0101
```

```
0011
```

```
------
```

```
0001
```

```
=1
```

Bitwise OR (|)

Rule:

At least one bit is **1**

```
0101

0011

------

0111

=7
```

Bitwise XOR (^)

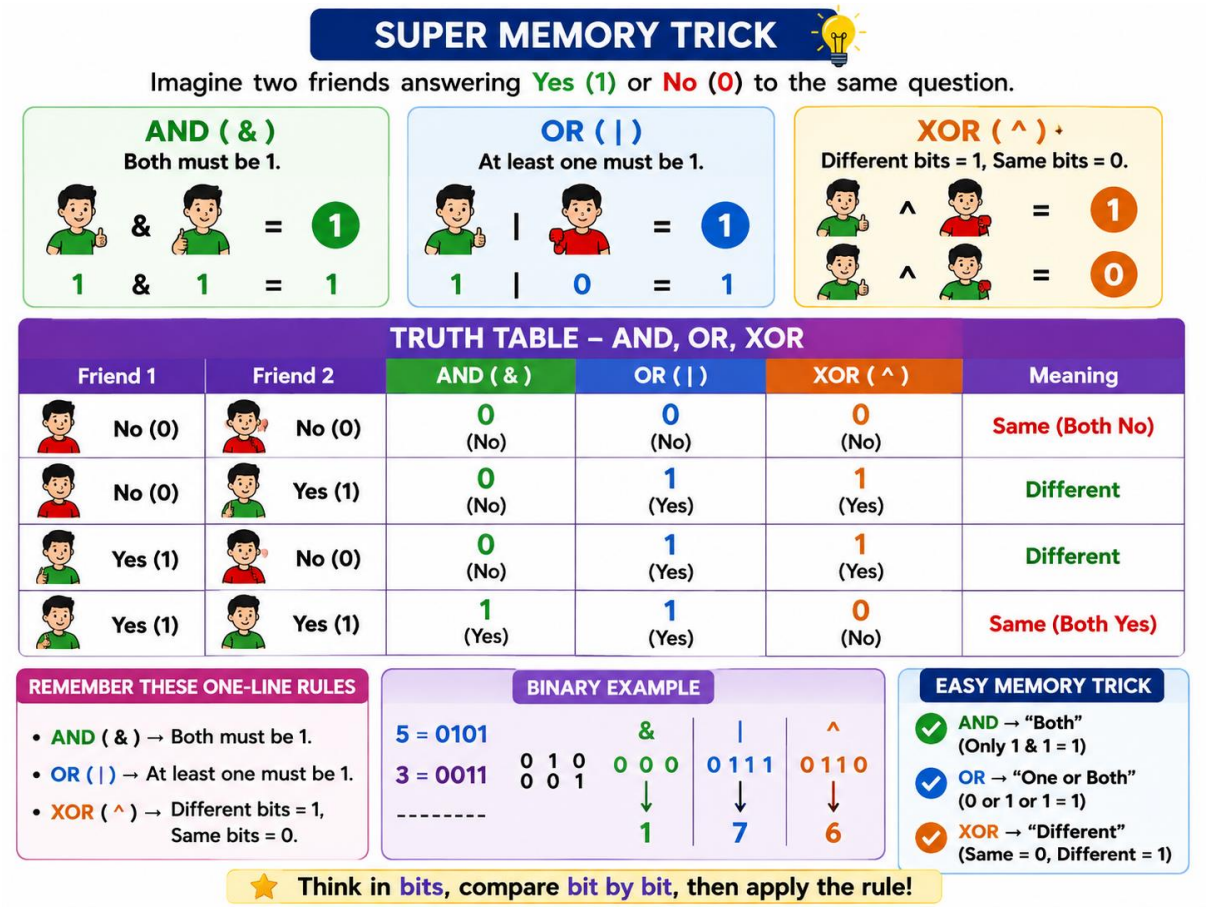
Rule:

Bits are **different**

```
0101

0011

------

0110

=6
```

Comparison Table

A	B	A & B	A | B	A ^ B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0


SUPER MEMORY TRICK
Imagine two friends answering Yes (1) or No (0) to the same question.
AND (&)
Both must be 1.
1 & 1 = 1
OR (|)
At least one must be 1.
1 | 0 = 1
XOR (^)
Different bits = 1, Same bits = 0.
1 ^ 0 = 1
1 ^ 1 = 0
TRUTH TABLE – AND, OR, XOR
Friend 1 | Friend 2 | AND (&) | OR (|) | XOR (^) | Meaning
No (0) | No (0) | 0 (No) | 0 (No) | 0 (No) | Same (Both No)
No (0) | Yes (1) | 0 (No) | 1 (Yes) | 1 (Yes) | Different
Yes (1) | No (0) | 0 (No) | 1 (Yes) | 1 (Yes) | Different
Yes (1) | Yes (1) | 1 (Yes) | 1 (Yes) | 0 (No) | Same (Both Yes)
REMEMBER THESE ONE-LINE RULES
• AND (&) → Both must be 1.
• OR (|) → At least one must be 1.
• XOR (^) → Different bits = 1, Same bits = 0.
BINARY EXAMPLE
5 = 0101
3 = 0011
0 1 0
0 0 1
& 0 0 0 → 1
| 0 1 1 1 → 7
^ 0 1 1 0 → 6
EASY MEMORY TRICK
AND → "Both" (Only 1 & 1 = 1)
OR → "One or Both" (0 or 1 or 1 = 1)
XOR → "Different" (Same = 0, Different = 1)
Think in bits, compare bit by bit, then apply the rule!