# Bits & data

C Reference

## Bits, binary & bitwise operators

A bit 位 is a single 0 or 1; numbers are stored in binary 二进制. Bitwise 按位 operators work on the bits directly: `&` AND, `|` OR, `^` XOR, `<<` shift left (×2 each step), `>>` shift right (÷2).

```c
#include <stdio.h>

int main(void) {
    int a = 12;              // 1100
    int b = 10;              // 1010
    printf("%d\n", a & b);   // 8   AND  -> 1000
    printf("%d\n", a | b);   // 14  OR   -> 1110
    printf("%d\n", a ^ b);   // 6   XOR  -> 0110
    printf("%d\n", a << 1);  // 24  left shift
    printf("%d\n", a >> 1);  // 6   right shift
    return 0;
}
```

Hexadecimal 十六进制 (base 16) writes binary compactly: one hex digit is exactly four bits. Write hex literals with `0x`; print with `%x`:

```c
#include <stdio.h>

int main(void) {
    printf("%d\n", 0xFF);   // 255  (0x marks a hex literal)
    printf("%x\n", 255);    // ff
    printf("%x\n", 12);     // c
    return 0;
}
```




Exam tip: bitwise ops work on bits of integers; << multiplies by 2, >> divides by 2

*Bitwise operators work on the bits of integers*

## Run-length encoding

Run-length encoding (RLE) is a simple compression 压缩 method: replace each run 游程 of repeated characters with a count and the character. It is lossless 无损 — the original is fully recoverable.

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

int main(void) {
    char *s = "aaabbc";
    int n = strlen(s);
    for (int i = 0; i < n; ) {
        char c = s[i];
        int run = 0;
        while (i < n && s[i] == c) { run++; i++; }
        printf("%d%c", run, c);     // 3a2b1c
    }
    printf("\n");
    return 0;
}
```

### Common mistakes

- `&` is bitwise AND and `&&` is logical AND — do not confuse them.
- A left shift `<< 1` doubles a value; a right shift `>> 1` halves it.
- Bit `n` has value `1 << n` (C has no `**` power operator); check a bit with `(x >> n) & 1`.