&	AND
|	OR
^	XOR
~	NOT
<<	Dịch bit sang trái
>>	Dịch bit sang phải

Các phép toán thao tác bit cơ bản

Phép toán AND (&)

Kết quả của phép AND sẽ là 1 nếu cả 2 toán hạng là 1. Nếu một trong hai toán hạng là 0 thì kết quả sẽ là 0, sau đây là bảng chân trị của phép AND:

A	**B**	**A & B**
0	0	0
1	0	0
0	1	0
1	1	1

Ví dụ phép AND giữa 2 số thập phân là 5 và 3:

```
0
1   0101 (5)
2 & 0011 (3)
3 = 0001 (1)
4
```

Minh họa với C++:

```
0
#include <iostream>

using namespace std;

int main()
{
    int a = 5, b = 3;
    cout << "Output = " << a&b;
    return 0;
}

```

Kết quả:

```
0
Output = 1

```

Phép toán OR (|)

Kết quả của phép OR sẽ là 1 nếu một trong hai toán hạng là 1. Trong C++, phép OR được ký hiệu là |. Bảng chân trị của phép OR

A	B	A | B
0	0	0
1	0	1
0	1	1
1	1	1

Ví dụ phép OR của 12 và 25

```
  00001100 (12)
| 00011001 (25)
= 00011101 (29)
```

Minh họa với C++:

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a = 12, b = 25;
    cout << "Output = " << a|b;
    return 0;
}
```

Kết quả:

```
Output = 29
```

Phép toán XOR (^)

Kết quả của phép XOR là 1 nếu 2 toán hạng có giá trị khác nhau.

A	B	A ^ B
0	0	0
1	0	1
0	1	1
1	1	0

```
  00011110 (30)
^ 00001001 (9)
= 00010111 (23)
```

Minh họa với C++:

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a = 30, b = 9;
    cout << "Output = " << a^b;
    return 0;
}
```

Kết quả:

Kết quả:

```
Output = 23
```

Ngoài ra, có 2 tính chất đặc biệt với phép XOR:

- A ^ A = 0 (1 toán hạng XOR với chính nó sẽ bằng 0)
- A ^ 0 = A (Bất kỳ toán hạng nào XOR với 0 đều bằng chính nó)

Phép toán NOT (~)

Toán tử NOT là toán tử 1 ngôi. Nó thay đổi toán hạng từ 0 sang 1 và ngược lại

A	**~A**
0	1
1	0

```
~ 00011110 (30)
= 11100001 (225)
```

Minh họa với C++:

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Output = " << ~30;
    return 0;
}
```

Kết quả:

```
Output = -31
```

Chúng ta thấy kết quả của ~30 là -31 thay vì 225, tại sao lại như vậy?

Để hiểu được điều này chúng ta sẽ nói qua một chút về bù 2 (2's complement):

Bù 2 của một số sẽ bằng bù 1 (~) của số đó cộng thêm 1

```
0
1 Số thập phân   Số nhị phân        Bù 2
2 0              00000000           -(11111111+1) = -00000000 = -0(Số thập phân)
3 1              00000001           -(11111110+1) = -11111111 = -256(Số thập phân)
4 12             00001100           -(11110011+1) = -11110100 = -244(Số thập phân)
5 225            11100001           -(00011110+1) = -00011111 = -31(Số thập phân)
6
```

Đối với bất kỳ số nguyên `n`, thì `~n` sẽ bằng `-(n+1)`. Vì `~n` được biểu diễn dưới dạng bù 2 và bù 2 của `~n` sẽ là `-(~(~n)+1)=-(n+1)`

Suy ra kết quả ở đầy sẽ là -31 thay vì 225 vì bù 1 của 30 (~30) là 225, bù 2 của 225 là -31.

Toán tử dịch bit

Dịch phải (>>)

Toán tử dịch phải bit sẽ dịch tất cả các bit sang phải bởi một số nhất định

```
0
1 212 = 11010100 (Số nhị phân)
2 212>>2 = 00110101 (Số nhị phân) (Dịch sang phải 2 bits)
3 212>>7 = 00000001 (Số nhị phân) (Dịch sang phải 7 bits)
4 212>>8 = 00000000 (Số nhị phân) (Dịch sang phải 8 bits)
5 212>>0 = 11010100 (Giữ nguyên)
6
```

Dịch trái bit (<<)

Toán tử dịch trái bit sẽ dịch tất cả các bit sang trái bởi một số nhất định

```
212 = 11010100 (Số nhị phân)
212<<1 = 11010100 (Số nhị phân) (Dịch sang trái 1 bít)
212<<0 = 11010100 (Giữ nguyên)
212<<4 = 11010100 (Số nhị phân) (Dịch sang trái 4 bít)
```

Minh họa với C++:

```cpp
#include <iostream>

using namespace std;

int main() {
    int num = 212, i;
    for(i = 0; i<=2; i++) {
        cout << "Dịch sang phải " << i << " bits: " << (num>>i) << "\n";
    }

    cout << "\n";

    for(i = 0; i<=2; i++) {
        cout << "Dịch sang trái " << i << " bits: " << (num<<i) << "\n";
    }
    return 0;
}
```

Kết quả:

```
Dịch sang phải 0 bits: 212
Dịch sang phải 1 bits: 106
Dịch sang phải 2 bits: 53

Dịch sang trái 0 bits: 212
Dịch sang trái 1 bits: 424
Dịch sang trái 2 bits: 848
```

Bài tập phép toán thao tác bit

Bạn có thể thực hành các bài tập liên quan tới phép toán thao tác bit (bitwise) trên nền tảng Luyện Code tại đây. Sau đây sẽ là 2 bài toán kinh điển có sự hỗ trợ của phép toán thao tác bit đi kèm hướng dẫn và lời giải tham khảo sử dụng C++.

Bài tập phép toán thao tác bit

Bạn có thể thực hành các bài tập liên quan tới phép toán thao tác bit (bitwise) trên nền tảng Luyện Code tại đây. Sau đây sẽ là 2 bài toán kinh điển có sự hỗ trợ của phép toán thao tác bit đi kèm hướng dẫn và lời giải tham khảo sử dụng C++.

Kiểm tra tính chẵn lẻ của một số

Đề bài: *Cho 1 số nguyên n, kiểm tra xem n là chẵn hay lẻ?*

Chúng ta sẽ thấy 1 điều rằng những số chẵn ở dạng nhị phân thì bit ngoài cùng bên phải sẽ luôn là bit 0, đối với số lẻ thì bit ngoài cùng bên phải sẽ luôn là 1. Dựa vào điều này ta có thể dễ dàng biết được tính chẵn lẻ bằng cách lấy số đó AND(&) với 1 nếu kết quả bằng 1 thì đó là số lẻ, ngược lại nếu bằng 0 sẽ là số chẵn. Ví dụ:

```
0
1   0100 (4) |   0011 (3)
2 & 0001 (1) | & 0001 (1)
3 = 0000 (0) | = 0001 (1)
4
```

Lời giải tham khảo với C++:

```
0
1  #include <iostream>
2
3  using namespce std;
4
5  int main() {
6      int n = 9;
7      if(n & 1 == 1) {
8          cout << "Lẻ";
9      } else {
10         cout << "Chẵn";
11     }
12 }
13
```

Tìm số xuất hiện 1 lần duy nhất trong mảng

Đề bài: *Cho 1 mảng các số nguyên, mỗi số trong mảng xuất hiện 2 lần, ngoại trừ 1 số xuất hiện đúng 1 lần, hãy tìm số xuất hiện đúng 1 lần đó?*

Đối với bài toán này chúng ta sẽ có khá nhiều cách giải, có thể sử dụng hash table, độ phức tạp thời gian sẽ là O(n) nhưng lại yêu cầu nhiều bộ nhớ hơn. Vậy ở đây chúng ta sẽ sử dụng các phép toán bit với độ phức tạp thời gian là O(n) và không gian là O(1).

Như mình đã đề cập ở phần XOR(^), phép XOR có 2 tính chất đặc biệt, và ý tưởng cho bài toán này là chúng ta chỉ cần XOR hết tất cả các phần tử trong mảng lại với nhau, 2 phần tử trùng nhau sẽ trả về về 0 và còn lại phần tử xuất hiện đúng 1 lần sẽ XOR với 0 và trả về phần tử đó.

```
0
1 Giả sử có mảng arr như sau
2 arr = [2, 3, 5, 4, 5, 3, 4]
3 Thực hiện XOR tất cả các phần tử với nhau:
4     2 ^ 3 ^ 5 ^ 4 ^ 5 ^ 3 ^ 4
5 <=> 2 ^ 3 ^ 3 ^ 4 ^ 4 ^ 5 ^ 5 (đổi vị trí cho dễ nhìn)
6 <=> 2 ^ 0 ^ 0 ^ 0 (sử dụng tính chất A ^ A = 0)
7 <=> 2 (sử dụng tính chất A ^ 0 = A)
8
```

Lời giải tham khảo với C++:

```
0
#include <iostream>

using namespace std;

int findUnique(int arr[], int n) {
    int result = 0;
    for(int i=0; i< n; i++) {
        result ^= arr[i];
    }
    return result;
}

int main() {
    int arr[] = {2, 3, 5, 4, 5, 3, 4};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Output = " << findUnique(arr, n);
    return 0;
}

```

Kết quả:

```
0
Output = 2

```