TOÁN TỬ BIT (BITWISE OPERATOR)

Giới thiệu về toán tử bit:

Toán tử bit là những toán tử được thực hiện trên các bit ở dạng biểu diễn nhị phân của các toán tử. Vì là các toán tử thực hiện trên bit nên tốc độ thực thi của toán tử bit rất nhanh.

CÁC TOÁN TỬ BIT

/01. Toán tử AND: &

/02. Toán tử OR: |

/03. Toán tử NOT: ~

/04. Toán tử XOR: ^

/05. Toán tử dịch trái: <<

/06. Toán tử dịch phải: >>

1. Toán tử AND: &

Giả sử ta có hai số không dấu a = 37 và b = 22. Hai số này có kiểu int vì thế nó sẽ có 32 bit biểu diễn.

a	00...00	1	0	0	1	0	1
b	00...00	0	1	0	1	1	0
a & b	00...00	0	0	0	1	0	0

```
#include <iostream>
using namespace std;
int main(){
    unsigned int a = 37, b = 22;
    unsigned int c = a & b;
    cout << c << endl;
}
```

OUTPUT

4

2. Toán tử OR: |

a	00...00	1	0	0	1	0	1
b	00...00	0	1	0	1	1	0
a | b	00...00	1	1	0	1	1	1

```cpp
#include <iostream>
using namespace std;

int main(){
    unsigned int a = 37, b = 22;
    unsigned int c = a | b;
    cout << c << endl;
}
```

OUTPUT

55

3. Toán tử NOT: ~

a	00…00	1	0	0	1	0	1
~a	11…11	0	1	1	0	1	0

```cpp
#include <iostream>
using namespace std;
int main(){
    unsigned int a = 37, b = 22;
    unsigned int c = ~a;
    cout << c << endl;
}
```

OUTPUT

4294967258

4. Toán tử XOR: ^

Bảng chân lý của cổng XOR

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

Cách nhớ bảng chân lý toán tử XOR :
Nếu số lượng bit 1 là số lẻ thì phép XOR có kết quả là 1.

4. Toán tử XOR: ^

a	00...00	1	0	0	1	0	1
b	00...00	0	1	0	1	1	0
a ^ b	00...00	1	1	0	0	1	1

```cpp
#include <iostream>
using namespace std;
int main(){
    unsigned int a = 37, b = 22;
    unsigned int c = a ^ b;
    cout << c << endl;
}
```

OUTPUT

51

4. Toán tử XOR: ^

Nếu bạn XOR một số với chính nó sẽ ra số 0. Còn XOR với số 0 sẽ ra chính nó

Ví dụ:

```
#include <iostream>
using namespace std;
int main(){
    unsigned int a = 37, b = 22;
    unsigned int c = a ^ a;
    cout << c << endl;
}
```

OUTPUT

0

4. Toán tử XOR: ^

Bài toán: Cho mảng số nguyên, trong mảng chỉ có duy nhất 1 số có tần suất lẻ, hãy tìm số đó

Hướng giải quyết: Ta dựa vào tính chất trên của toán tử bit XOR. Vì các số có tần suất chẵn XOR với nhau sẽ bằng 0 hết nên ta chỉ cần XOR mọi số trong mảng sẽ tìm ra số có tần suất lẻ duy nhất.

```
#include <iostream>
using namespace std;
int main(){
    int a[] = {1, 1, 1, 1, 2, 2, 3, 3, 4, 4, 4};
    int ans = 0;
    for(int i = 0; i < 11; i++)
        ans ^= a[i];
    cout << ans << endl;
}
```

OUTPUT

4

5. Toán tử dịch trái: <<

Toán tử dịch trái tương ứng với việc bạn dịch các bit của số ban đầu sang trái 1 vị trí và bổ sung thêm 1 bit 0 ở cuối. Vì thế phép dịch trái tương đương với việc bạn nhân số ban đầu với 2.

Ví dụ: a = 1001 khi dịch trái 1 bit a sẽ thành 10010

```cpp
#include <iostream>
using namespace std;
int main(){
    int a = 10;
    int b = (a << 1);
    cout << b << endl;
    int c = (a << 3);
    cout << c << endl;
}
```

OUTPUT

```
20
80
```

Nếu bạn dịch trái K bit thì số a sẽ thành số a * 2^K

6. Toán tử dịch phải: >>

Ngược lại với dịch trái, nếu bạn dịch phải K bit thì bạn sẽ chia số ban đầu với 2^K

```
#include <iostream>
using namespace std;
int main(){
    int a = 100;
    int b = (a >> 1);
    cout << b << endl;
    int c = (a >> 3);
    cout << c << endl;
}
```

OUTPUT

```
50
12
```

7. Một vài trick với toán tử bit:

Kiểm tra số lẻ bằng cách & số ban đầu với 1

```
#include <iostream>
using namespace std;
int main(){
    int a = 11;
    if(a & 1){
        cout << “Le”;
    }
    else{
        cout << “Chan”;
    }
}
```

OUTPUT

Le

Tính nhanh 2^K bằng cách cho 1 dịch trái K bit

```
#include <iostream>
using namespace std;
int main(){
    cout << (1 << 10) << endl;
}
```

OUTPUT

1024

7. Một vài trick với toán tử bit:

Sinh mọi tập con của 1 tập sử dụng bitmasking

```
#include <iostream>
using namespace std;
int main(){
    int a[] = {1, 2, 3};
    for(int i = 0; i < (1 << 3); i++){
        cout << "{ “;
        for(int j = 0; j < 3; j++){
            if(i & (1 << j)){
                cout << a[j] << ‘ ‘;
            }
        }
        cout << “ }\n”;
    }
}
```

OUTPUT

```
{}
{1}
{2}
{1 2}
{3}
{1 3}
{2 3}
{1 2 3}
```