# Bitwise Operation

errantProgrammer

October 5, 2024

# Contents

**1 Bitwise Operation** **3**
- 1.1 AND & . . . 3
  - 1.1.1 Codigo Example . . . 3
- 1.2 OR | . . . 3
  - 1.2.1 Codigo Example . . . 3
- 1.3 XOR ^ . . . 3
  - 1.3.1 Codigo Example . . . 4
- 1.4 NOT ∼ . . . 4
  - 1.4.1 Codigo Example . . . 4
- 1.5 Left Shift ≪ . . . 4
  - 1.5.1 Codigo Example . . . 4
- 1.6 Right Shift ≫ . . . 5
  - 1.6.1 Codigo Example . . . 5
- 1.7 Complemento a 2 . . . 5
- 1.8 Problemas . . . 5

# 1 Bitwise Operation

## 1.1 AND &

| P | Q | P & Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

### 1.1.1 Codigo Example

```
6     int a = 7, b = 4;
7     int p = a & b;
8     cout <<"a & b = " << p << endl; // p = 4
```

$$\begin{array}{rr} & 111_2 \\ \& & 100_2 \\ \hline & 100_2 = 4 \end{array}$$

## 1.2 OR |

| P | Q | P \|Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

### 1.2.1 Codigo Example

```
10    int c = 7, d = 4;
11    int q = c | d;
12    cout << "c | d = " << q << endl; // q = 7
13    // XOR
```

$$\begin{array}{rr} & 111_2 \\ | & 100_2 \\ \hline & 111_2 = 7 \end{array}$$

## 1.3 XOR ^

| P | Q | P ^Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

### 1.3.1 Codigo Example

```
6     int a = 7, b = 4;
7     int p = a & b;
8     cout <<"a & b = " << p << endl; // p = 4
```

$$\begin{array}{r} 111_2 \\ \hat{} \quad 100_2 \\ \hline 011_2 = 3 \end{array}$$

## 1.4 NOT $\sim$

El not lo que hace nos brinda el complemento a 2, del número, no confundir con el negador lógico "!".

| P | $\sim$ P |
|---|---|
| 0 | 1 |
| 1 | 0 |

### 1.4.1 Codigo Example

```
6     int a = 7, b = 4;
7     int p = a & b;
8     cout <<"a & b = " << p << endl; // p = 4
```

$$\begin{array}{r} \sim \quad 1001_2 \\ \hline 0110_2 = 6 \end{array}$$

## 1.5 Left Shift $\ll$

| P | Q | P $\ll$ Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

### 1.5.1 Codigo Example

```
6     int a = 7, b = 4;
7     int p = a & b;
8     cout <<"a & b = " << p << endl; // p = 4
```

$$\begin{array}{r} 111_2 \\ \ll \quad 2 \\ \hline 011_2 = 3 \end{array}$$

| P | Q | P $\gg$ Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

## 1.6 Right Shift $\gg$

### 1.6.1 Codigo Example

```
6       int a = 7, b = 4;
7       int p = a & b;
8       cout <<"a & b = " << p << endl; // p = 4
```

$$\begin{array}{rr} & 111_2 \\ \gg & 2 \\ \hline & 011_2 = 3 \end{array}$$

## 1.7 Complemento a 2

El **complemento a 2** de un número es la forma de como representamos números negativos de un número, para calcular su valor existen multiples formas:

- $C_2(N) = 2^n - N$
- $C_2(N) = C_1(N) + 1$

Pero la forma más sencilla que tenemos de calcularlo, es representando el número a binario, y desde el *bit menos significativo*, avanzamos hacia la izquierda hasta encontrar el primer 1, y a partir de este, invertimos ceros y unos.

## 1.8 Problemas

Resolución de algunos problemas:

```
1   #include <bits/stdc++.h>
2
3   int main(){
4       int n;
5       std::cin >> n; //cantidad de angulos
6       std::vector<int> angles(n);
7       for (int i = 0; i < n; i++) std::cin >> angles[i];
8
9       for (int mask = 0;mask < (1 << n);mask++){ // recorremos la mascara
10          int total = 0;// suma total de los angulos
11          for (int i = 0; i < n; i++){ // solo vamos a utilizar los n primeros bits
12              if(mask & (1 << i)) // verificamos si el i-esimo bit esta prendido
13                  total += angles[i];
14              else
```

```
                total -= angles[i];
        }
        if(total % 360 == 0){
            std::cout << "YES";
            return 0;//se acaba el programa
        }
    }

    std::cout << "NO";


    return 0;
}
```