



BITWISE OPERATORS



Bitwise operation

A	B	$\sim A$	$A \& B$	$A B$	$A \wedge B$
1	1	0	1	1	0
1	0	0	0	1	1
0	1	1	0	1	1
0	0	1	0	0	0



Bitwise operation

- $A = 27 = 11011_2$
- $B = 83 = 1010011_2$
- $A \& B = 19, A | B = 91, A \wedge B = 73$

0 0 1 1 0 1 1	0 0 1 1 0 1 1	0 0 1 1 0 1 1
& 1 0 1 0 0 1 1	1 0 1 0 0 1 1	^ 1 0 1 0 0 1 1
-----	-----	-----
0 0 1 0 0 1 1	1 0 1 1 0 1 1	1 0 0 1 0 0 0



NOT

- $A = 83 = 1010011_2$
- $\sim A = 10101100_2$ (8 bit)
- $\sim A = 11111111\ 11111111\ 11111111\ 10101100_2$ (32 bit)



Shift Left

- $1 \ll 0 = 1$
- $1 \ll 1 = 2 = 10_2$
- $1 \ll 2 = 4 = 100_2$
- $1 \ll 3 = 8 = 1000_2$
- $1 \ll 4 = 16 = 10000_2$
- $3 \ll 3 = 24 = 11000_2$
- $5 \ll 10 = 5120 = 10100000000000_2$



Shift Right

- $1 \gg 0 = 1$
- $1 \gg 1 = 0 = 0_2$
- $(10_{10}) \ 1010_2 \gg 1 = 5 = 101_2$
- $(10_{10}) \ 1010_2 \gg 2 = 2 = 10_2$
- $(10_{10}) \ 1010_2 \gg 3 = 1 = 1_2$
- $30 \gg 1 = 15 = 1111_2$
- $1024 \gg 10 = 1 = 1_2$



Bitwise Operators

- $A \ll B = A * 2^B$
- $A \gg B = A / 2^B$
- $(A+B) / 2 = (A + B) \gg 1$
- FIND ODDS
 - $\text{if}(N \% 2 == 1)$
 - $\text{if}(N \& 1)$



Packing/unpacking

```
int pack(char a, char b, char c, char d)
{
    int    p = a;
    p = (p << 8) | b;
    p = (p << 8) | c;
    p = (p << 8) | d;
    return p;
}
```

```
char unpack(int p, int k)
{
    int    n = k * 8;
    unsigned mask = 255;
    mask <=< n;
    return ((p & mask) >> n);
}
```




Packing/unpacking

int 형

a	b	c	d
---	---	---	---

```
#include <limits.h> // CHAR_BIT = 8
int pack(char a, char b, char c, char d)
{
    int p = a;
    p = (p << CHAR_BIT) | b;
    p = (p << CHAR_BIT) | c;
    p = (p << CHAR_BIT) | d;
    return p;
}
```

p	0	0	0	a
	0	0	a	b
	0	a	b	c
	a	b	c	d



Packing/unpacking

01100001

01100010

01100011

01100100

11111111

```
char unpack(int p, int k)
{
    int n = k * 8;
    unsigned mask = 255;
    mask <<= n;
    return ((p & mask) >> n);
}
```

k = 0

d



Packing/unpacking

01100001

01100010

01100011

01100100

11111111

```
char unpack(int p, int k)
{
    int n = k * 8;
    unsigned mask = 255;
    mask <<= n;
    return ((p & mask) >> n);
}
```

k = 0

d

k = 1

c



Packing/unpacking

01100001

01100010

01100011

01100100

11111111

```
char unpack(int p, int k)
{
    int n = k * 8;
    unsigned mask = 255;
    mask <<= n;
    return ((p & mask) >> n);
}
```

k = 0

d

k = 1

c

k = 2

b



Packing/unpacking

01100001

01100010

01100011

01100100

11111111

```
char unpack(int p, int k)
{
    int n = k * 8;
    unsigned mask = 255;
    mask <<= n;
    return ((p & mask) >> n);
}
```

k = 0

d

k = 1

c

k = 2

b

k = 3

a



Packing/unpacking

```
char unpack(int p, int k)
{
    int n = k * 8;
    unsigned mask = 255;
    mask <= n;
    return ((p & mask) >> n);
}
```

C:\Windows\system32\cmd.exe

abcd = 01100001 01100010 01100011 01100100

abcd = dcba



BitMask

- $\{1, 3, 4, 5, 9\}$
- $570 = 2^1 + 2^3 + 2^4 + 2^5 + 2^9$
- $= (1000111010_2)$
- Checking if number 0 is included
 - ✓ $570 \& 2^0 = 570 \& (1 \ll 0) = 0$
- Checking if number 1 is included
 - ✓ $570 \& 2^1 = 570 \& (1 \ll 1) = 2$
- Checking if number 2 is included
 - ✓ $570 \& 2^2 = 570 \& (1 \ll 2) = 0$
- Checking if number 8 is included
 - ✓ $570 \& 2^3 = 570 \& (1 \ll 3) = 8$



BitMask

- $\{1,3,4,5,9\} = 570 (1000111010_2)$
- Adding 1
 - ✓ $570 | 2^1 = 570 | (1 \ll 1) = 570 (1000111010_2)$
- Adding 2
 - ✓ $570 \& 2^2 = 570 | (1 \ll 3) = 574 (1000111110_2)$
- Adding 3
 - ✓ $570 \& 2^3 = 570 | (1 \ll 4) = 570 (1000111010_2)$
- Adding 4
 - ✓ $570 \& 2^4 = 570 | (1 \ll 5) = 570 (1000111010_2)$



BitMask

- $\{1,3,4,5,9\} = 570$
- Removing 1
 - ✓ $570 \& \sim 2^1 = 570 \& \sim(1 \ll 1) = 568(1000111000_2)$
- Removing 2
 - ✓ $570 \& \sim 2^2 = 570 \& \sim(1 \ll 2) = 570(1000111010_2)$
- Removing 3
 - ✓ $570 \& \sim 2^3 = 570 \& \sim(1 \ll 3) = 562(1000110010_2)$
- Removing 4
 - ✓ $562 \& \sim 2^4 = 562 \& \sim(1 \ll 5) = 546(1000101010_2)$



BitMask

➤ Adding i

✓ $s | (1 \ll i)$

➤ Cheking i

✓ $s \& (1 \ll i)$

➤ Removing i

✓ $s \& \sim(1 \ll i)$

➤ Toggling i

✓ $s \wedge \sim(1 \ll i)$



Swap (using XOR operator)

- `int a = 3, b = 7; //a = 0011, b = 0111`
- `a = a ^ b; //a = 4`
- `a = a ^ b; //a = 3`



Swap (using XOR operators)

- `int a = 3, b = 7; //a = 0011, b = 0111`
- `a = a ^ b; //a = 4`
- `b = b ^ a; //b = 3`
- `a = a ^ b; //a = 7`
- **Short Coding**
- `a ^= b ^= a ^= b;`