

Logical Functions

Logical Functions					
A	B	A AND B	A OR B	A XOR B	NOT A
1	1	1	1	0	0
1	1	1	1	0	0
0	0	0	0	0	1
0	0	0	0	0	1
0	0	0	0	0	1
0	0	0	0	0	1
1	0	0	1	1	0
1	0	0	1	1	0

Logical Operators In C

Logical Operators		
Operator	Description	Example (A = 1 , B = 0)
&&	Logical AND operator - The condition result becomes true only if both operands are non-zero.	(A && B) → false
	Logical OR Operator - The condition result becomes true If any of the two operands is non-zero.	(A B) → true
!	Logical NOT Operator - It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) → true

Bitwise Operators in C

Bitwise Operators		
Operator	Description	Example A: 0110 0001 B: 0000 0100
&	Binary AND - copies a bit to the result if it exists in both operands.	(A & B) → 0000 0000
	Binary OR - copies a bit if it exists in either operand.	(A B) → 0110 0101
^	Binary XOR: <ul style="list-style-type: none"> ✓ copies the bit if it is set in one operand but not both. ✓ equivalent to adding two bits and discarding the carry. ✓ the result is zero only when we have two zeroes or two ones. ✓ can be used to toggle the bits between 1 and 0. 	(A ^ B) → 0110 0101
~	Binary Ones Complement (unary) – flip the bits.	(~A) → 1001 1110
<<	Binary Left Shift: $a \ll N = a * 2^N$ <ul style="list-style-type: none"> ✓ Shift the bits of the left operand n-times to the left as specified by the right operand. ✓ shift left an unsigned int by K is equivalent to multiplying by 2^K (zero will be shifted in from least significant end) ✓ Shift left on signed values is ok but overflow occurs when the most significant bit changes values (from 0 to 1 or 1 to 0). 	(A << 2) → 1000 0100
>>	Binary Right Shift: $a \gg N = a / (2^N)$ <ul style="list-style-type: none"> ✓ Shift the bits of the left operand n-times to the right as specified by the right operand. ✓ shift right an unsigned int by K is equivalent to devid by 2^K (dose not work for negative values) ✓ When shifting to the right for unsigned int, bits fall off the least significant end, and 0's are shifted in from the most significant end. However, with signed int, then right shifting depends on the compiler. 	(A >> 2) → 0001 1000

Bitwise assignment operators in C

Bitwise assignment operators		
Operator	Description	Example A: 0110 0001 B: 0000 0100
&=	bitwise AND assignment.	(A &= B) → A = 0000 0000
=	bitwise inclusive OR assignment	(A = B) → A = 0110 0101
^=	bitwise exclusive OR assignment	(A ^= B) → A = 0110 0101
<<=	left shift assignment	(A <<= 2) → A = 1000 0100
>>=	right shift assignment	(A >>= 2) → A = 0001 1000

Bitwise vs Logical Operators

- bitwise operators are equivalent to logical operators in that they have the same truth tables.
 - logical operators treat each operand as having only one value either true or false.
 - bitwise operators treat each bit of an operand as an independent value.
 - logical operators consider zero value as false and nonzero values as true.
 - in contrast of bitwise, logical operators perform short-circuit evaluation.
-
- `!=` has the same truth table as `^` but unlike the true logical operators, by itself `!=` is not strictly speaking a logical operator. This is because a logical operator must treat any nonzero value the same. To be used as a logical operator `!=` requires that operands be normalized first. A logical not applied to both operands won't change the truth table that results but will ensure all nonzero values are converted to the same value before comparison. This works because `!` on a zero always results in a one and `!` on any nonzero value always results in a zero.

Bitwise vs Logical Operators	
Bitwise	Logical
<code>A & B</code>	<code>A && B</code>
<code>A B</code>	<code>A B</code>
<code>A ^ B</code>	<code>A != B</code>
<code>~A</code>	<code>!A</code>

Example 1

```
#include <stdio.h>
int main() {
    unsigned char a = 'a';           → 0110 0001
    unsigned char b = 4;             → 0000 0100
    unsigned char c = 0;

    c = a & b;                       → C: 0000 0000 = 0
    printf("%d\n", c);
    c = a | b;                       → C: 0110 0101 = 101
    printf("%d\n", c);
    c = a ^ b;                       → C: 0110 0101 = 101
    printf("%d\n", c);
    c = ~a;                          → C: 1001 1110 = 158
    printf("%d\n", c);
    c = a << 2;                      → C: 1000 0100 = 132
    printf("%d\n", c);
    c = a >> 2;                      → C: 0001 1000 = 24
    printf("%d\n", c);
    return 1;
}
```

Output:

0
101
101
158
132
24

```
#include <stdio.h>
int main() {
    unsigned int a = 0xffffffff;
    unsigned int b = 4;
    unsigned int c = 0;

    c = a & b;
    printf("%-10d %-10u %-10X\n", c, c, c);
    c = a | b;
    printf("%-10d %-10u %-10X\n", c, c, c);
    c = a ^ b;
    printf("%-10d %-10u %-10X\n", c, c, c);
    c = ~a;
    printf("%-10d %-10u %-10X\n", c, c, c);
    c = a << 2;
    printf("%-10d %-10u %-10X\n", c, c, c);
    c = a >> 2;
    printf("%-10d %-10u %-10X\n", c, c, c);
    return 1;
}
```

Output:

4	4	4
-1	4294967295	FFFFFFFF
-5	4294967291	FFFFFFFB
0	0	0
-4	4294967292	FFFFFFFC
1073741823	1073741823	3FFFFFFF

Example 3

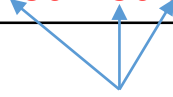
```
#include <stdio.h>
int main()
{
    unsigned char a = 255;
    unsigned char b = ~a;
    unsigned int c = 255;
    unsigned int d = ~c;

    printf("%X %X %X %X %X %X\n", a, ~a, ~255, b, c, d);
    printf("%u %u %u %u %u %u\n", a, ~a, ~255, b, c, d);
    printf("%d %d %d %d %d %d\n", a, ~a, ~255, b, c, d);

    return 1;
}
```

Output:

```
FF FFFFFFF0 FFFFFFF0 0 FF FFFFFFF0
255 4294967040 4294967040 0 255 4294967040
255 -256 -256 0 255 -256
```



Why?

?

- ✓ Integer promotion on bitwise operation!
- ✓ casting

Write a program to check the count of ones in a given unsigned long number.

```
#include <stdio.h>
int main() {
    unsigned int a = 0x01234567;
    int i,c=0;
    const int INT_SIZE = sizeof(unsigned int)*8;

    for(i=0; i<INT_SIZE ;i++)
    {
        ((1u<<i) & a) ? c++ : 0 ;
    }

    printf("%d\n", c);

    return 1;
}
```

version1

* The loop is executed as the size of unsigned long.

```
#include <stdio.h>
int main() {
    unsigned int a = 0x01234567;
    int c=0;

    while(a)
    {
        a = a & (a-1);
        c++;
    }

    printf("%d\n", c);

    return 1;
}
```

version2

* The loop is executed, exactly as the count of ones in 'a'.

Given an integer number 'a' write a program which:

1. Reads bit #i.
2. Writes 1 in bit #i.
3. Writes 0 in bit #i.
4. Swaps 0 <--> 1 in bit #i

* 0 <= i <= 31

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

```
int main()
```

```
{
```

```
int a = 4, i = 3;
```

```
a & 1u << i ? putchar('1') : putchar('0');
```

```
a = a | 1u << i;  
printf("\n%d\n", a);
```

```
a = a & ~(1u << i);  
printf("%d\n", a);
```

```
a & 1u << i ? (a &= ~(1 << i)) : (a |= 1 << i);  
printf("%d\n", a);
```

```
return 1;
```

```
}
```

1. Reads bit #i.

2. Writes 1 in bit #i.
a |= 1u << i

3. Writes 0 in bit #i.
a &= ~(1u << i)
a & 1u << i ? a = a ^ 1u << i : 1

4. Swaps 0 <--> 1 in bit #i
a ^= 1u << i

Output:

0
12
4
12

Given an integer number
check if it is even or odd
using bitwise operator

```
#include <stdio.h>
int main()
{
    int num = 122;

    if(num & 1u)
    {
        printf("%d is odd.", num);
    }
    else
    {
        printf("%d is even.", num);
    }

    return 1;
}
```

Swapping 2 numbers using bitwise operators

```
#include <stdio.h>
int main() {
    int a = 2, b = 3;

    a ^= b;
    b ^= a;
    a ^= b;
    printf("%d %d\n",a ,b);

    return 1;
}
```

Write a function showBin which
receives an integer and prints its
binary show

```
#include <stdio.h>
#define INT_SIZE sizeof(int) * 8
void showBin(int a)
{
    short i = INT_SIZE;
    char bin[INT_SIZE + 1];
    bin[INT_SIZE + 1] = '\0';
    while(i != 0) {
        bin[--i] = (a & 1u ? '1' : '0');
        a >>= 1;
    }
    printf("%s\n", bin);
}

int main()
{
    showBin(4);
    return 1;
}
```

version1

```
#include <stdio.h>
void showBin (int a)
{
    short i = sizeof(int)*8 -1;
    while (i >= 0)
    {
        (a & (1u << i--)) ? putchar('1') : putchar('0');
    }
    putchar('\n');
}

int main()
{
    showBin(97);
    return 1;
}
```

version2

Below program calculates the
sum of 2 unsigned integers using
 \wedge , $\&$, \ll

```
#include <stdio.h>
unsigned sum (unsigned int x, unsigned int y) {
    unsigned int carry;

    while (y != 0)
    {
        carry = x & y;
        x = x ^ y;
        y = carry << 1u;
    }
    return x;
}

int main() {
    printf("%u\n", sum(2u,255u) );
    return 1;
}
```

version1

Iterate till there is no carry

carry now contains
common set bits of x and y

Sum of bits of x and y
where at least one of the
bits is not set

Carry is shifted by one so
that adding it to x gives the
required sum

Recursion version

```
#include <stdio.h>
unsigned sum (unsigned int x, unsigned int y) {
    if (y == 0)
        return x;
    else
        return sum( x ^ y, (x & y) << 1u);
}

int main() {
    printf("%u\n", sum(2u,255u) );
    return 1;
}
```

version2

Given an array of integers
write a program that checks
the count of negative
numbers.

```
#include <stdio.h>
int main()
{
    int arr[] = {-1,-2,-3,1,2,3,-4};
    int i, c=0;

    for(i=0; i<7; i++)
    {
        if((arr[i] & (1u<<31)) != 0)
        {
            c++;
        }
    }

    printf("%d\n", c);

    return 1;
}
```

Given an array of integers write a program that checks the count of positive even numbers.

```
#include <stdio.h>
int main()
{
    int arr[] = {0,-1,-2,-3,1,2,3,-4,4};
    int i, c=0;

    for(i=0; i<9; i++)
    {
        if(((arr[i] & (1u<<31)) == 0) && ((arr[i]&1u) == 0))
        {
            c++;
        }
    }

    printf("%d\n", c);

    return 1;
}
```


Given an unsigned number 'a' and an int number 'n' write the below functions:

zeroNBitsFromLeft(a, n) : zero all n bits from the left.

zeroNBitsFromRight(a, n) : zero all n bits from the right.

```
#include <stdio.h>
unsigned int zeroNBitsFromLeft(unsigned int, int);
unsigned int zeroNBitsFromRight(unsigned int, int);
int main() {
    printf("%x\n", zeroNBitsFromLeft(0xffffffff, 31));
    printf("%x\n", zeroNBitsFromRight(0xffffffff, 31));
    return 1;
}

unsigned int zeroNBitsFromLeft(unsigned int a, int n) {
    const int INT_SIZE = sizeof(int)*8;
    int i;
    for (i=INT_SIZE-1; i>=INT_SIZE-n && i>=0 ; i--) {
        a = ~(1u<<i)&a;
    }
    return a;
}

unsigned int zeroNBitsFromRight(unsigned int a, int n) {
    const int INT_SIZE = sizeof(int)*8;
    int i;
    for (i=0; i<=n-1 && i<INT_SIZE ; i++) {
        a = ~(1u<<i)&a;
    }
    return a;
}
```

Bitwise Operations - Jazmawi Shadi

Output:

1
80000000

Given unsigned int 'a' and 'b' (b = 2ⁿ) - using bitwise operations only, write a program which calculates a*b, a/b and a%b

$$\log_b(a) = \frac{\log_x(a)}{\log_x(b)}$$

← Change of base rule

log(..) : base 'e'
Log2(..) : base 2

```
#include <stdio.h>
#include <math.h>
double log2(double a) {
    return log(a)/log(2);
}
unsigned int mult(unsigned int a, unsigned int b) {
    int n = (int)log2((double)b);
    return a<<n;
}
unsigned int div(unsigned int a, unsigned int b) {
    int n = (int)log2((double)b);
    return a>>n;
}
unsigned int mod(unsigned int a, unsigned int b) {
    const int INT_SIZE = sizeof(int)*8;
    int n = (int)log2((double)b);
    return (a<<(INT_SIZE-n))>>(INT_SIZE-n);
}
int main() {
    printf("%u %u %u\n", mult(255u,8u), div(256u,8u), mod(257u,8u));
    return 1;
}
```

b = 2ⁿ

→ a*b = a*2ⁿ → a<<n

→ a/b = a/2ⁿ → a>>n

Output:

2040 32 1

Given a char number 'a' – based on bitwise operations only, write a function reverse(a) which returns the reversed bits of 'a'.

```
#include <stdio.h>
int main() {
    char a = 0x11, b=0;
    int i = sizeof(char)*8-1, j=0;

    while(i>=0)
    {
        (a & (1u<<i--)) ? (b|=1u<<j++) : (b&=~(1u<<j++));
    }

    printf("%x %x\n", a, b);

    return 1;
}
```

Output:
11 ffffff88

find if the binary show of a given unsigned char
'a', is a sub set of a given unsigned long 'b'

0x499602D2 = 1234567890

0100-1001-1001-0110-0000-0010-1101-0010

0xCB = 1100 1011

```
#include <stdio.h>
#define LONG_SIZE sizeof(unsigned long)*8
#define CHAR_SIZE sizeof(char)*8
int main() {
    unsigned char a=0xcb;
    unsigned long b=1234567890l;
    int i=0,found=0;

    while(i<=(LONG_SIZE-CHAR_SIZE) && !found) {
        b=1234567890l;
        b=(b<<i)>>1l;
        b=(~(1l<<(LONG_SIZE-1)))&b;
        b=b>>(LONG_SIZE-CHAR_SIZE-1);
        (a==b) ? (found=1):(found=0);
        i++;
    }
    if(found)
        printf("Begin at %d\n",i);
    else
        printf("not found\n");

    return 1;
}
```

Output:
Begin at 8

END