C BITWISE OPERATORS

It does bitwise manipulation

Bitwise

AND → &	LEFT SHIFT → <<
OR → |	RIGHT SHIFT → >>
NOT → ~	XOR → ^

→ AND op. takes 2 bits at a time & performs AND operation.

→ AND & OR are binary operators. It takes two ng.s & perform bitwise AND

→ Result of AND is 1 when both bits are 1

```
7 →   0 1 1 1
4 → & 0 1 0 0
    -------
4 ←   0 1 0 0
```

T.T.

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

7 & 4 = 4

Same for OR

```
7 →   0 1 1 1
4 → | 0 1 0 0
    -------
      0 1 1 1
```

7 | 4 = 7

NOT is a unary operator, it basically complements each bit

7 ⟶ ~ 0 1 1 1

8 ⟵ 1 0 0 0

~7 = 8

DIFFERENCE BETN LOGICAL & BITWISE OP.

```
int main()
{
char x=1, y=2; // x=1 (0000 0001),
                  y=2 (0000 0010) */

if (x & y)  // 1 & 2 = 0 (0000 0000)
   printf("Result of x & y is 1");

if (x && y)  // 1 && 2 = True && True = True = 1
   printf("Result of x && y is 1");

   return 0;
}
```

O/p: Result of x && y is 1

LEFT SHIFT OPERATOR

First operand << second operand

↓ Whose bits get left shifted

↳ Decides the number of places to shift the bits

→ When bits are shifted left trailing positions are filled with zeros.

```
#include <stdio.h>

int main() {
char var = 5; /* Note 3 in binary =
                 0000 0011 */
printf("%d", var << 1);
return 0;
}
```

O/P = 6

How left shift works?

var << 1

↓

left shift by one position

var = 3

3 = 0000 0011

↓

0000 0110

↓

0000 0110

Trailing pos filled with zero

= 6

Imp. points

① left shifting is equivalent to multiplication by $2^{\text{right Operand}}$

e.g. var = 3

var << 1 o/p = 6 [3×2^1]

var << 4 o/p = 48 [3×2^4]

RIGHT SHIFT OPERATOR

First operand >> second operand

Whose bits get right shifted — Decides no. of places to shift the bits

→ When bits are shifted right then leading pos's are filled with zeros

```
Int main() {
char var = 3; // Note: 3 in binary = 0000 0011
printf("%d", var >> 1);
return 0;
}
```

o/p 1

How right shift works?

var >> 1

var = 3

3 = 0000 0011

Right shift by one posn : 0000 0001

Leading posn filled with zero → 0000 0001 = 1

Imp. pts.

Right shift is equivalent to division by $2^{\text{right operand}}$

E.g. var = 3

var >> 1 o/p : 1 [3/2¹]

var = 32

var >> 4 o/p : 2 [32/2⁴]

BITWISE XOR

Inclusive OR

- Either A or B is 1 or both are 1, then the o/p is 1
- Including both

Exclusive OR

- Either A is 1 or B is 1 then the o/p is 1 but when both A & B are 1 the o/p is 0.
- Excluding both

Date
Page

* Bitwise X-OR (^) is a binary operator. It takes 2 ops & performs bitwise XOR.
* Result of XOR is 1 when 2 bits are diff. otherwise result is 0.

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

7 → 0 1 1 1

4 → ^ 0 1 0 0

3 ← 0 0 1 1

7 ^ 4 = 3

E.g.

```
int main() {
    int a = 4, b = 3;
    a = a ^ b;
    b = a ^ b;
    a = a ^ b;

    printf("After XOR, a = %d b = %d", a, b);
    return 0;
}
```

(1) a = a ^ b

4 → 0 1 0 0

3 ^ → 0 0 1 1

a = 7 ← 0 1 1 1

(2) b = a ^ b = 7 ^ 3

7 → 0 1 1 1

3 ^ → 0 0 1 1

b = 4 ← 0 1 0 0

(3) a = a ^ b = 7 ^ 4

7 → 0 1 1 1

4 ^ → 0 1 0 0

a = 3 ← 0 0 1 1

o/p - a = 3 b = 4

<math.h> function

Funct^{n}	Type	Purpose
abs(i)	int	Return the absolute value of i.
ceil(d)	double	Round up to the next integer value (smallest integer that is ~~less~~ greater than or equal to d)
cos(d)	double	Return the cosine of d
cosh(d)	double	Return the hyperbolic cosine of d
exp(d)	double	Raise e to the power d

fabs(d)	double	Return the absolute value of d
floor(d)	double	Round down to the next integer value (the largest integer that does not exceed d)
fmod(d1,d2)	double	Return the remainder
getchar()	int	Enter a character from standard input device
log(d)	double	Return the natural log of d
pow(d1, d2)	double	Return d1 raised to d2 power
putchar(c)	int	Send a character to standard output device
rand()	int	Return a random positive integer
sin(d)	double	Return the sine of d
sqrt(d)	double	Return the square root of d
srand(u)	void	Initialize the random number generator
tan(d)	double	Return the tangent of d
toascii(c)	int	Convert value of arguments to ASCII
tolower(c)	int	Convert letter to lowercase
toupper	int	Convert letter to uppercase

Date
Page

Break

Syntax:

```
break;
```

Effect

When the break statement is executed inside a loop-statement, the loop statement is terminated immediately.

The execution of program will continue with the statement following the loop statement.

schematically

```
while (loop-continuation-condition)
{
  statement1
  statement2
  ....
  break;
  ....
}
                ! Execution proceeds to here
statement  <-- statement following the while loop
```

Continue

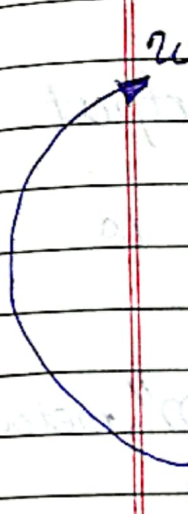
Syntax

```
continue;
```

Effect:

→ When the continue statement is executed inside a loop-statement, the program will skip over the remainder of the loop-body to the end of the loop.

→ In the case of a while loop, when the program reaches end of the loop, the program will jump back to the testing of the loop-continuation-condition.



```
while (loop-continuation-condition)
  {
    st. 1
    st. 2
    .....
    continue;
    .....
  }

statement ← statement foll. the while loop
```

```
#include <stdio.h>
int main()
  {
  int j;
  for (j=0; j<=8; j++)
    { if (j==4)
        { /* The continue statement is
encountered when * the value of j is equal
  to 4. */
        continue;
    }
```

```
/* This print statement won't execute
for the * loop iteration where j==4
because in that case * this statement
would be skipped. */
        printf("%d", j);
    }
    return 0;
}
```

Formatting Numbers in Program Output

- Field Width (the no. of columns used to display a value)

```
Printf("Results: %3d meters = %4d ft.\n", meters, feet);
```

Results: □21 meters = □□68 ft.

Value	Format	Disp. Out.	Value	Format	D.O.
234	%4d	□234	-234	%4d	-234
234	%5d	□□234	-234	%5d	□-234
234	%6d	□□□234	-234	%6d	□□-234
234	%1d	234	-234	%2d	-234

Date
Page

Value	Format	O.O.
3.14159	%.5.2f	□3.14
11	%.3.2f	3.14
11	%.5.3f	3.142
.1234	%.4.2f	0.12
~~=.006~~		
-.006	%.8.3f	□□-0.006
-.006	%.3f	-0.006

%.3d prints an integer that is atleast 3 spaces long

Handshake problem

$$h(n) = h(n-1) + n - 1$$