Lesson 34 Bitwise Operator AND

Programming branches into two parts:

- The commands.

- The way of using these commands.

In this course, we study the **C language commands**.

In order to create a functional program that has no errors and works quickly, we must learn **algorithms**.

The concept covered in this lesson is hard to put into practice, however, the algorithm course will cover it soon.

Typically, a number is made up of **Bits** and **Bytes**.

We already know that any number is represented as **0's** and **1's.**

int x;

x has **32 bits** consisting of **0's** and **1's.**

For example:

Number **5** (1 0 1) is stored as follows:

00000000000000000000000000000101

Every **int** has **32 bits** total, divided into 4 parts. Each part has **8 bits** and is called a **byte**.

- **int =4 byte**
- **int = 32 bit**
- **Byte = 8 bit**
- - > one binary digit is called a bit.

Bitwise

As the name implies, it works on the bit level.

int x = 5, y = 6;

int z = x & y;

- what does y & x = z mean?

64	32	16	8	4	2	1	
0	0	0	0	1	0	1	→ x in binary
0	0	0	0	1	1	0	→ y in binary
0	0	0	0	1	0	0	→ x & y

The & operator means:

- If **both** bits are **1**, then the result of OR is 1 (true).
- If only **one** of the two bits is **1**, then the result of OR is 0 (false).
- If **none** of the two bits is **1**, then the result of OR is 0 (false).

As a result, **z = 4.**

if(true && true)

Sometimes we use the **AND operator (&&)** in if statements.

It can only return true if **both** of the two conditions are true.

In the Bitwise operator, however, we use the (&) operator that works on the binary equivalent of decimal numbers bit by bit instead of the number as a whole.

int x = 11, y = 3;

int z = x & y;

64	32	16	8	4	2	1
0	0	0	1	0	1	1
0	0	0	0	0	1	1
0	0	0	0	0	1	1

As a result, z = 4.

True & True = True

True & false = False