**Operators**

Table of Contents

[Types of Operators 2](#_Toc65497373)

[Precedence 7](#_Toc65497374)

## Types of Operators

(Page 52, Chapter 03 – ANSCII Book)

Operators perform functions. Operands are data on which these functions are performed.

There are 2 categories of operators: Unary, which work on 1 operand, and binary, which work on 2.

There are 8 types of operators.

1. Arithmetic – These are binary operators. Using integer operands will give integer results, and using double operands will give double results. They include addition (+), subtraction (-), multiplication (\*), division (/) and modulus (%).

2. Relational – These are binary operators. They check the relationship between two operands. They include less than (<), greater than (>), greater than or equal to (>=), less than or equal to (<=), equal to (==) and not equal to (!=). The statements themselves are always correct in the C language, even if they return a false value. E.g. 3>4 is a correct statement and will return a false value.

3. Logical – These include and (&& - binary), or (|| - binary) and not (! – unary). The last of these, not (!) will reverse the result of a statement if it is placed in front of the statement.

4. Assignment – These are binary. They include =, +=, -=, \*=, /= and %=.

5. Increment/Decrement – These are unary. They include ++ and <--. While assigning values, b = ++a means the value of a increases by 1, and b is assigned to this value and b = a++ means b is assigned to the value of a and then the value of a increases.

6. Conditional – These include if statements, else statements and else if statements.

if (true)  
{  
 execute code;  
}  
else if (true)  
{  
 execute code;  
}  
else *// if everything above was false*{  
 execute code;  
}

C

This can also be written as: exp1? exp2:exp3

E.g.

int a = 3;  
int b = 3>7? ++a: --a;

C

7. Bitwise – These can manipulate the actual binary code that is used to store data. They include &, |, ^, << and >>.

a = 3, b = 4;

C

In binary code, 3 = 0011 (0+0+21+20) and 4 = 0100.

So, the statement a&b is checked as,

0011

0100

0000 - false

For a|b,

0011

0100

0111 - true

For a^b (XOR) For XOR,

0011 1 – 1 false

0100 1 – 0 true

0111 - true 0 – 0 false

<< and >> literally shift the binary code left and right respectively.

a = 3 0011

a << 1 0110 0 + 22 + 21 + 20 = 7 a = 7

a >> 2 0001 the extra bit is lost and cannot be brought back

Example use of bitwise functions:

Create an array of 10 characters to check if a specific index position contains data or not:

Method 1: int array [10] *160 bits memory*

Use for loop to cycle through each index position. *Slow*

Method 2: int a = 3; *16 bits memory*

(binary code for a = 0000000110 so only positions 2 and 3 have data)

int b = 2\*n; (Enter position n to be checked.

a&b;

(This statement checks if position 1 for both integers have data)

0000000110

0000000001

0000000000 - false

So, there is no data in the first position. *Super fast*

8. Special - These are symbols that have a special meaning in the C language. These include ,, & and \*.

, separates data [as in printf ("%d %d", a, b)];

& stores addresses [as in int a = &b (a stores memory address of b)]

\* is a pointer [as in \*c]

## Precedence

(Page 73, Table 3.8, ANSCII Book)

This refers to the order of operations in an equation. Some operators perform their functions before others (like division before addition according to BODMAS). It is less confusing to just use brackets instead.