Operators

OPERATOR

An operator performs a specific type of operation on one, two or three operands and produces the result. They can be categorised in two types:

Based on number of operands	Type of operation
• Unary • Binary • Ternary	• Logical operator • Arithmetic operator • Relational operator • Assignment operator • Bitwise operator

Unary operator

- **+** Represent positive number — `int i = +10;`
- **-** Represent negative number — `int i = -10;`
- **++** Increment the value by 1 — `int i = 10++;` - post-increment; `int i = ++10;` - pre-increment

Post-Increment

Input	Output

int a = 10;
int b = a++;
S.o.p (b);
S.o.p (a);
	10
	11

It will first assign the value and then it will increment the value of a by 1

Pre-Increment

Input	Output

int a = 10;
int b = ++a;
S.o.p (b);
S.o.p (a);
	11
	11

It will first increment the value by 1 and then it will assign the value to b

- **--** Decrement the value by 1 — `int i = 10--;` - post-decrement; `int i = --10;` - pre-decrement

Same for decrement operator as shown of Increment operator

Binary Arithmetic Operator

+	Addition between two values	20 + 10
-	Subtraction between two values	20 - 10
*****	Multiplication between two values	20 * 10
/	Division between two values	20 / 10
%	Gets reminder of the division	20 % 10

Assignment operator

=	Assigning RHS to LHS	int a = b
+=	Assigning LHS + RHS to LHS	int a += 10
-=	Assigning LHS - RHS to LHS	int a -= 10
***=**	Assigning LHS * RHS to LHS	int a *= 10
/=	Assigning LHS / RHS to LHS	int a /= 10
%=	Assigning LHS % RHS to LHS	int a %= 10

Relational operator

==	Is LHS equal to RHS?	a == b
!=	Is LHS not equal to RHS	a != b
>	Is LHS greater than to RHS	a > b
<	Is LHS less than to LHS * RHS	a < b
>=	Is LHS greater than equal to LHS / RHS	a >= b
<=	Is LHS less than equal to LHS % RHS	a <= b

Logical operator

- **&** AND *-in bitwise operator-*
- **&&** If both are true result is true `(false && (5 / 0 == 0)); //false`
- **|** OR *-in bitwise operator-*
- **||** If any one is true result is true `(true || (5 / 0 == 0)); //true`
- **!** If result is true it will reverse it `(true ^ true); // false`
- **^** XOR *-in bitwise operator-*

Bitwise Operator

This takes place in the binary form that is 1 and 0 of the given number

- **&**

If both is one then output is one

100 - 4
&
101 - 5
100 = output 4

```
int i = 4;
int i2 = 5;
S.o.p(i & i2);
```

4

- **|**

If any one is one then output is one

100 - 4
|
101 - 5
101 = output 5

```
int i = 4;
int i2 = 5;
S.o.p(i | i2);
```

5

- **^**

If both are same then output is 0

100 - 4
^
101 - 5
001 = output 1

```
int i = 4;
int i2 = 5;
S.o.p(i ^ i2);
```

1

- **~**

It makes 1 a 0 and 0 a 1

0010 - 2
~
1101 = output -3

```
int i = 2;
S.o.p(~i);
```

-3

- **>>**

right shift

The operator shifts all bits towards the right by a certain number of specified bits by adding 0 from left. *E.g., right shift by 1*

0101 - 5
>>
0010 = output 2

```
int i = 5;
S.o.p(i >> 1);
```

2

• >>>	unsigned right shift	the operator '>>>' denotes unsigned right shift operator and always fill 0 irrespective of the sign of the number	
	int i = -5; S.o.p(i >>> 1);	2147483645	
• <<	left shift	The operator shifts all bits towards the left by a certain number of specified bits by adding 0 from right. *E.g., left shift by 1*	0101 - 5 << 1010 = output 10
	int i = 5; S.o.p(i << 1);	10	

Ternary Operator

We will be learning ternary operator when we will be learning ***if – else – if*** statement