1. What is an Operator?

An **operator** is a symbol used to perform an operation on one or more values or variables.

Example

```
int a = 10;
int b = 5;

int result = a + b;
```

Here:

- `a` and `b` are **operands**.
- `+` is an **operator**.
- `a + b` is an **expression**.
- The result is `15`.

---

2. What is an Expression?

An **expression** is a combination of values, variables, and operators that produces a result.

Examples

```
a + b
```

```
marks >= 50
```

```
age >= 18 && age <= 60
```

Each of these expressions produces a result.

---

3. Types of Operators Based on Number of Operands

Operators can be classified according to the number of operands they work with.

3.1 Unary Operators

A **unary operator** works with **one operand**.

Examples

```
x++;
```

```
-x;
```

```
!isStudent;
```

Only one variable/value is involved.

Common Unary Operators

Operator	Meaning	Example
`++`	Increment	`x++`
`--`	Decrement	`x--`
`-`	Negative	`-x`
`!`	Logical NOT	`!x`

3.2 Binary Operators

A **binary operator** works with **two operands**.

Example

```
a + b
```

Here:

- `a` → first operand
- `+` → operator
- `b` → second operand

Other examples:

```
a - b
a * b
a > b
a == b
```

Remember

Unary → one operand

```
x++;
```

Binary → two operands

```
x + y;
```

---

4. Types of C++ Operators

The main operators we will study are:

1. Arithmetic Operators
2. Assignment Operator
3. Compound Assignment Operators
4. Increment and Decrement Operators
5. Relational Operators
6. Logical Operators

---

5. Arithmetic Operators

Arithmetic operators are used to perform **mathematical calculations**.

Operator	Name	Example
`+`	Addition	`a + b`
`-`	Subtraction	`a - b`
`*`	Multiplication	`a * b`
`/`	Division	`a / b`
`%`	Modulus/Remainder	`a % b`

Example

```
int a = 10;
int b = 3;

cout << a + b;   // 13
cout << a - b;   // 7
cout << a * b;   // 30
cout << a / b;   // 3
cout << a % b;   // 1
```

Important Point: / Division Operator

The / operator is used for division.

The result depends on the data type of the operands.

1. Both Operands are int

If both operands are integers, C++ performs **integer division**.

```
int a = 10;
int b = 3;

cout << a / b;
```

Output:

```
3
```

Although mathematically:

```
10 ÷ 3 = 3.3333...
```

C++ gives:

```
3
```

because both a and b are int.

> **Important:** The decimal part is removed when integer division is performed.

---

2. One Operand is float

If at least one operand is a float, the division produces a decimal result.

```
float a = 10;
int b = 3;

cout << a / b;
```

Output:

```
3.33333
```

Here:

```
float / int → decimal result
```

---

3. Both Operands are `float`

```
float a = 10;
float b = 3;

cout << a / b;
```

Output:

```
3.33333
```

Here:

```
float / float → decimal result
```

---

4. Using `double`

`double` can also be used for decimal calculations.

```
double a = 10;
double b = 3;

cout << a / b;
```

Output:

```
3.33333
```

Remember

Operand Types	Example	Result
`int / int`	`10 / 3`	`3`
`float / int`	`10.0f / 3`	`3.33333`
`int / float`	`10 / 3.0f`	`3.33333`
`float / float`	`10.0f / 3.0f`	`3.33333`

Operand Types	Example	Result
double / double	10.0 / 3.0	3.33333

Important Rule

> **If both operands are `int`, the result of `/` is integer division. If at least one operand is a floating-point type (`float` or `double`), the result can contain decimal values.**

Important Point: `%` Remainder Operator

The `%` operator gives the **remainder after integer division**.

```
int a = 10;
int b = 3;

cout << a % b;
```

Output:

```
1
```

Because:

```
10 ÷ 3 = 3 remainder 1
```

Therefore:

```
10 / 3 = 3
10 % 3 = 1
```

Another Example

```
int a = 20;
int b = 6;

cout << a / b << endl;
cout << a % b << endl;
```

Output:

```
3
2
```

Because:

```
20 ÷ 6 = 3 remainder 2
```

Remember

```
/  → gives the quotient
%  → gives the remainder
```

For example:

```
17 / 5 = 3
17 % 5 = 2
```

6. Assignment Operator

The assignment operator is:

```
=
```

It is used to **assign a value to a variable**.

Example

```
int age = 20;
```

This means:

> Store the value `20` in the variable `age`.

Another example:

```
int x = 10;

x = 25;
```

Now:

```
x = 25
```

Important

Do not confuse:

```
=
```

with:

```
==
```

= means **assignment**.

== means **comparison**.

Example:

```
x = 10;    // Assign 10 to x
x == 10;   // Check whether x is 10
```

---

7. Compound Assignment Operators

Compound assignment operators provide a shorter way to update a variable.

Example

Instead of:

```
x = x + 5;
```

we can write:

```
x += 5;
```

Both have the same effect.

Common Compound Assignment Operators

Operator	Example	Equivalent To
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5

Operator	Example	Equivalent To
%=	x %= 5	x = x % 5

Example

```
int marks = 70;

marks += 5;
```

Now:

```
marks = 75
```

---

8. Increment Operator ++

The increment operator increases a value by **1**.

```
int x = 5;

x++;
```

Now:

```
x = 6
```

It is equivalent to:

```
x = x + 1;
```

There are two forms:

```
x++;
```

Post-increment

and:

```
++x;
```

Pre-increment

Both increase the value by 1. The difference becomes important when the operator is used inside a larger expression.

---

9. Decrement Operator --

The decrement operator decreases a value by **1**.

```
int x = 5;

x--;
```

Now:

```
x = 4
```

It is equivalent to:

```
x = x - 1;
```

There are two forms:

```
x--;
```

Post-decrement

and:

```
--x;
```

Pre-decrement

---

10. Relational Operators

Relational operators are used to **compare two values**.

The result of a comparison is either:

```
true
```

or:

```
false
```

Relational Operators

Operator	Meaning	Example
`>`	Greater than	`a > b`
`<`	Less than	`a < b`
`>=`	Greater than or equal to	`a >= b`
`<=`	Less than or equal to	`a <= b`
`==`	Equal to	`a == b`
`!=`	Not equal to	`a != b`

Example

```
int marks = 75;

cout << (marks >= 50);
```

The expression is:

```
marks >= 50
```

Since 75 is greater than 50, the result is:

```
true
```

---

11. Logical Operators

Logical operators are used to **combine or reverse conditions**.

There are three main logical operators:

Operator	Name	Meaning
`&&`	AND	Both conditions must be true
`` ` ``		`` ` `` OR At least one condition must be true
`!`	NOT	Reverses the condition

11.1 AND Operator &&

The && operator returns true when **both conditions are true**.

Example

```
age >= 18 && marks >= 50
```

This means:

> Age must be 18 or above **AND** marks must be 50 or above.

11.2 OR Operator ||

The || operator returns true when **at least one condition is true**.

Example

```
day == 6 || day == 7
```

This means:

> Day is 6 **OR** day is 7.

11.3 NOT Operator !

The ! operator reverses a Boolean value or condition.

Example

```
bool isStudent = true;

cout << !isStudent;
```

The result becomes:

```
false
```

Because ! changes `true` to `false` and `false` to `true`.

---

12. Operators in `if` Statements

Operators are commonly used to create conditions in `if` statements.

Example 1

```
int marks = 75;

if (marks >= 50)
{
    cout << "Pass";
}
```

Here:

```
marks >= 50
```

is a **condition/expression** using a relational operator.

Example 2

```
int age = 20;
int marks = 75;

if (age >= 18 && marks >= 50)
{
    cout << "Eligible";
}
```

Here:

- `>=` → Relational operator
- `&&` → Logical operator
- `age >= 18` → First condition
- `marks >= 50` → Second condition

---

13. Operator Summary

Category	Operators	Purpose
Arithmetic	`+ - * / %`	Mathematical calculations
Assignment	`=`	Assign a value
Compound Assignment	`+= -= *= /= %=`	Update a variable
Increment	`++`	Increase by 1
Decrement	`--`	Decrease by 1

Category	Operators	Purpose
Relational	`> < >= <= == !=`	Compare values
Logical	`&&	!`	Combine or reverse conditions
Unary	`++ -- - !`	Work with one operand
Binary	`+ - * / > < ==` etc.	Work with two operands

14. Quick Revision

Operator

A symbol used to perform an operation.

Operand

A value or variable on which an operator works.

Expression

A combination of values, variables, and operators that produces a result.

Unary

Works with **one operand**.

```
x++;
```

Binary

Works with **two operands**.

```
x + y;
```

Arithmetic

Used for calculations.

```
+  -  *  /  %
```

Relational

Used for comparison.

```
>  <  >=  <=  ==  !=
```

Logical

Used to combine or reverse conditions.

```
&&  ||  !
```

Assignment

Used to assign a value.

```
=
```

Compound Assignment

Used to update a value in a shorter way.

```
+=  -=  *=  /=  %=
```

Increment / Decrement

Used to increase or decrease a value by 1.

```
++  --
```