**C++ operator**

***C++ has 6 types of Operators:***

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bitwise Operators
5. Assignment Operators
6. Other Operators

**1. Arithmetic Operators:** (+, -, \*, /, %, ++, --).

These operators are used to perform arithmetic/mathematical operations on operands.

1. **Unary Operators**: (++, --)

Operators that operate or work with a single operand are unary operators.

1. **Binary Operators**: (+, -, \*, /, %)

Operators that operate or work with two operands are binary operators.

**2. Relational Operators:** (==, >=, <=,! =)

These are used for the comparison of the values of two operands.

**3. Logical Operators: (&&, ||,!)**

Logical Operators are used to combine two or more conditions

**4. Bitwise Operators: (&, |,^,~,<<,>>)**

The Bitwise operators are used to perform bit-level operations on the operands

1. The **& (bitwise AND)** - The result of AND is 1 only if both bits are 1.

2. The **| (bitwise OR) - T**he result of OR is 1 if any of the two bits is 1.

3. The **^ (bitwise XOR) -**The result of XOR is 1 if the two bits are different.

4. The **<< (left shift)** and **>>(Right shift)** -**The left-shift and right-shift operators are equivalent to multiplication and division by 2 respectively.**

5.The **~ (bitwise NOT)** - takes one number and inverts all bits of it.

**5. Assignment Operators:** (+=,-=,\*=,/=,%=)

Assignment operators are used to assign value to a variable.

**6. Other Operators**:

**a.**[**sizeof operator**](https://www.geeksforgeeks.org/sizeof-operator-c/): It is a compile-time unary operator which can be used to compute the size of its operand

Ex. sizeof(char)

**b.**[**Comma Operator**](https://www.geeksforgeeks.org/comna-in-c-and-c/):  Comma acts as both operator and separator.

Ex: int a=10, b=21

void fun (10,21)

**c.**[**Conditional Operator**](https://www.geeksforgeeks.org/cc-ternary-operator-some-interesting-observations/): (Ternary Operator)

*syntax:* exp1? exp2: exp3

Ex: string result = (marks >= 40)? "passed”: "failed";

**d.** [**dot (.)**](https://www.geeksforgeeks.org/dot-operator-in-c-c/)**and**[**arrow (->)**](https://www.geeksforgeeks.org/arrow-operator-in-c-c-with-examples/)**Operators**

**-**The dot operator is applied to the actual object.

-The arrow operator is used with a pointer to an object.

Ex.  Struct Point p1= {1,2};

p1.x; p1.y

**e.**[**Cast Operator:**](https://www.geeksforgeeks.org/typecasting-in-c/) Casting operators convert one data type to another.

Ex. Int a=34, b=24;

Float div= (float)a/b;

**f.**[**&, \* Operator:**](https://www.geeksforgeeks.org/bitwise-operators-in-c-cpp/)

-Pointer operator & returns the address of a variable..

-Pointer operator \* is a pointer to a variable.

Ex: int x;

Int \*p = &x;

**Type Conversion in C++**

A type cast is basically a conversion from one type to another.

There are two types :

* + 1. **Implicit Type Conversion** Also known as ‘automatic type conversion’.

**-**Done by the compiler on its own

bool -> char -> short int -> int ->unsigned int -> long -> unsigned -> long long -> float -> double -> long double

Ex:

**int** x = 10;

**char** y = 'a';

x = x + y;

**float** z = x + 1.0;

* + 1. **Explicit Type Conversion**: This process is also called type casting and it is user-defined.

Ex:

**double** x = 1.2;

**int** sum = (**int**)x + 1;

**What is Memory Management?**

Memory management is a process of managing computer memory, assigning the memory space to the programs to improve the overall system performance.

-In [C language](https://www.javatpoint.com/c-programming-language-tutorial), we use the **malloc()** or **calloc()** functions to allocate the memory dynamically at run time. and free() function is used to deallocate the dynamically allocated memory.

-but C++ also defines unary operators such as **new** and **delete** to perform the same tasks.

### **New operator :**

### A **new** operator is used to create the object. When the object is created by using the new operator, then the object will exist until we explicitly use the delete operator to delete the object

**Syntax**: pointer variable = **new** data-type

**Ex :** int \*p;  p = new int;

We can also assign the values by using new operator.

**Syntax**: pointer variable = **new** data-type(value);

**Ex :** int \*p = new int[45];

**Delete operator :**

A **Delete** operator is release the memory. When **delete** is used to deallocate memory for a C++ class object

Ex: delete[] p;

**Scope Resolution Operator**

scope resolution operator is **::**

1. **To access a global variable when there is a local variable with same name**

**int** x;  // Global x

**int** main()

{

**int** x = 10; // Local x

  cout << "Value of global x is " << ::x;

  cout << "\nValue of local x is " << x;

**return** 0;

}

1. **To define a function outside a class.**

**class** A

{

**public**:

   // Only declaration

**void** fun();

};

// Definition outside class using ::

**void** A::fun()

{

   cout << "fun() called";

}

**int** main()

{

   A a;

   a.fun();

**return** 0;

}

1. **To access a class’s static variables.**

**class** Test

{

**static** **int** x;

**public**:

**static** **int** y;

    // Local parameter 'a' hides class member

    // 'a', but we can access it using ::

**void** func(**int** x)

    {

       // We can access class's static variable

       // even if there is a local variable

       cout << "Value of static x is " << Test::x;

       cout << "\nValue of local x is " << x;

    }

};

1. **In case of multiple Inheritance**

**class** A

{

**protected**:

**int** x;

**public**:

    A() { x = 10; }

};

**class** B

{

**protected**:

**int** x;

**public**:

    B() { x = 20; }

};

**class** C: **public** A, **public** B

{

**public**:

**void** fun()

   {

      cout << "A's x is " << A::x;

      cout << "\nB's x is " << B::x;

   }

};

1. **For namespace**

**int** main(){

    std::cout << "Hello" << std::endl;

}

1. **Refer to a class inside another class:**

**class** outside

{

**public**:

**int** x;

**class** inside

      {

**public**:

**int** x;

**static** **int** y;

**int** foo();

      };

};

**int** outside::inside::y = 5;