Operators in C++:

An operator is a symbol that operates on a value to perform specific mathematical or logical computations. They form the foundation of any programming Language.

- ❖ Unary operators
  - Increment - pre increment , post increment
  - Decrement - pre decrement , post decrement

- ❖ Binary operators
  - Arithmetic Operator
  - Relational Operators
  - Logical Operators
  - Assignment operators
  - Bitwise Operators

Unary operators:

Unary operators in C++ are those operators that work on a single value (operand)

Program:

```
#include <iostream>
using namespace std;

int main() {

    int a=5;
    cout<<(++a)<<endl; //pre increment - First increment by 1 and then use
    //Now the value of a is 6
    cout<<"The value of a After pre increment : "<<a<<endl<<endl;

    cout<<(a++)<<endl; //post increment - first use it and then increment by 1
    //now the value of a is 7
    cout<<"The value of a After post increment : "<<a<<endl<<endl;

    cout<<(--a)<<endl; //pre decrement - first decrement by 1 and then use;
    //Now the value of a is 6
    cout<<"The value of a After pre decrement : "<<a<<endl<<endl;

    cout<<(a--)<<endl; //post decrement - first use it and then decrement by 1
    //now the value of a is 5
    cout<<"The value of a After post decrement : "<<a<<endl<<endl;
}
```

Output:

PS E:\C++ PROGRAMMES> g++ main.cpp

PS E:\C++ PROGRAMMES> ./a.exe

6

The value of a After pre increment : 6

6

The value of a After post increment : 7

6

The value of a After pre decrement : 6

6

The value of a After post decrement : 5

Binary operator:

Operators that work with two operands: They take two values (operands) and produce a new value based on a specific operation.

Arithmetic operators :

Arithmetic Operators in C++ are used to perform arithmetic or mathematical operations on the operands. For example, '+' is used for addition, '–' is used for subtraction, '*' is used for multiplication, etc.

Program :

```
#include <iostream>
using namespace std;

int main() {
    //Binary operators - Arithmetic operators

    int a =10;
    int b=5;

    cout<<"Addition : "<<a+b<<endl;
    cout<<"subtraction : "<<a-b<<endl;
    cout<<"Multiplication : "<<a*b<<endl;
    cout<<"Division : "<<a/b<<endl;
    cout<<"Modulo Division : "<<a%b<<endl;
```

```
}
```

Output :

PS E:\C++ PROGRAMMES> g++ main.cpp

PS E:\C++ PROGRAMMES> ./a.exe

Addition : 15

subtraction : 5

Multiplication : 50

Division : 2

Modulo Division : 0

Relational Operators:

C++ Relational operators are used to compare two values or expressions, and based on this comparison, it returns a boolean value (either true or false) as the result. Generally false is represented as 0 and true is represented as any non-zero value (mostly 1).

1. Equal to: ==
2. Not equal to: !=
3. Less than: <
4. Greater than: >
5. Less than or equal to: <=
6. Greater than or equal to: >=

Program :

```cpp
#include <iostream>
using namespace std;

int main() {
    //Binary operators - Relational operators

    int a =10;
    int b=5;

    cout<<(a>b)<<endl; //Greaterthan operator
    cout<<(a<b)<<endl; //Lessthan operator
    cout<<(a>=b)<<endl; //Greaterthan or Equal to operator
    cout<<(a<=b)<<endl; //Lessthan or Equal to operator
    cout<<(a==b)<<endl; //Equal to operator
    cout<<(a!=b)<<endl; //Not Equal to operator

}
```

Output:

PS E:\C++ PROGRAMMES> g++ main.cpp

```
PS E:\C++ PROGRAMMES> ./a.exe
1
0
1
0
0
1
```

Logical Operators :

In C++ programming languages, logical operators are symbols that allow you to combine or modify conditions to make logical evaluations. They are used to perform logical operations on boolean values (true or false).

1. Logical AND (&&) Operator
2. Logical OR (||) Operator
3. Logical NOT (!) Operator

1. Logical AND Operator (&&)

The C++ logical AND operator (&&) is a binary operator that returns true if both of its operands are true. Otherwise, it returns false. Here's the truth table for the AND operator:

Operand 1	Operand 2	Result
true	true	**true**
true	false	**false**
false	true	**false**
false	false	**false**

Program :

```cpp
      #include <iostream>

using namespace std;

int main() {
    //Binary operators - Logical operators

    bool cond1 = true;
    bool cond2 = true;
    bool cond3 = false;
```

```
    //Logical AND (&&)

    if(cond1 && cond2 && cond3){
        cout<<"All conditions are true"<<endl;
    }
    else{
        cout<<"All conditions are not true"<<endl;
    }

}
```

Output:

PS E:\C++ PROGRAMMES> g++ main.cpp

PS E:\C++ PROGRAMMES> ./a.exe

All conditions are not true

2. Logical OR Operator (||)

The C++ logical OR operator (||) is a binary operator that returns true if at least one of its operands is true. It returns false only when both operands are false. Here's the truth table for the OR operator:

Operand 1	Operand 2	Result
true	true	**true**
true	false	**true**
false	true	**true**
false	false	**false**

Program:

```
#include <iostream>
using namespace std;

int main() {
    //Binary operators - Logical operators

    bool cond1 = (10==5);
    bool cond2 = (5<5);
    bool cond3 = (2!=2);

    //Logical OR (||)
```

```
    if(cond1 || cond2 || cond3){
        cout<<"Atleast one condition is true"<<endl;
    }
    else{
        cout<<"All conditions are false"<<endl;
    }
}
```

Output:

PS E:\C++ PROGRAMMES> g++ main.cpp

PS E:\C++ PROGRAMMES> ./a.exe

All conditions are false

3. Logical NOT Operator (!)

The C++ logical NOT operator (!) is a unary operator that is used to negate the value of a condition. It returns true if the condition is false, and false if the condition is true. Here's the truth table for the NOT operator:

Operand 1	Result
true	false
false	true

Program:

```
#include <iostream>
using namespace std;

int main() {
    //Binary operators - Logical operators

    bool condition = true;

    cout<<!condition<<endl;
}
```

Output:

PS E:\C++ PROGRAMMES> g++ main.cpp

PS E:\C++ PROGRAMMES> ./a.exe

0

Bitwise Operators:

Bitwise Operators are the operators that are used to perform operations on the bit level on the integers. While performing this operation integers are considered as sequences of binary digits. In C++, we have various types of Bitwise Operators.

1. Bitwise AND (&)
2. Bitwise OR (|)
3. Bitwise XOR (^)
4. Bitwise NOT (~)
5. Left Shift (<<)
6. Right Shift (>>)

The & (bitwise AND) in C takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.

The | (bitwise OR) in C takes two numbers as operands and does OR on every bit of two numbers. The result of OR is 1 if any of the two bits is 1.

The ^ (bitwise XOR) in C takes two numbers as operands and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.

The << (left shift) in C takes two numbers, the left shifts the bits of the first operand, and the second operand decides the number of places to shift.

The >> (right shift) in C takes two numbers, right shifts the bits of the first operand, and the second operand decides the number of places to shift.

The ~ (bitwise NOT) in C takes one number and inverts all bits of it.

X	Y	X & Y	X | Y	X ^ Y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Program :

```
#include <iostream>
using namespace std;
```

```cpp
int main() {
    //Binary operators - Bitwise operators

    int a = 5;
    int b =10;

    cout<<(a&b)<<endl;
    cout<<(a|b)<<endl;
    cout<<(a^b)<<endl;
    cout<<(~a)<<endl;
    cout<<(a<<2)<<endl;
    cout<<(a>>2)<<endl;
}
```

Output :

PS E:\C++ PROGRAMMES> g++ main.cpp

PS E:\C++ PROGRAMMES> ./a.exe

0

15

15

-6

20

1

Assignment operators:

Assignment operators are operators used in programming to store a value in a variable. They perform two tasks:

Calculate a value: They carry out a specific operation involving the variable's current value and a new value.

Assign the result: They store the calculated value back into the same variable, updating its contents.

1. = (simple assignment)
2. += (addition assignment)
3. -= (subtraction assignment)
4. *= (multiplication assignment)
5. /= (division assignment)
6. %= (modulo assignment)
7. &= (bitwise AND assignment)
8. |= (bitwise OR assignment)
9. ^= (bitwise XOR assignment)
10. <<= (left shift assignment)
11. >>= (right shift assignment)

Program :

```cpp
#include <iostream>
using namespace std;

int main() {
    //Binary operators - Assignment operators


    int a=5; //assignment operator
    cout<<a<<endl;

    a +=3; // addition asssignment
    cout<<a<<endl;

    a -=3; // subtraction asssignment
    cout<<a<<endl;

    a *=3; //multiplication asssignment
    cout<<a<<endl;

    a /=3; //division asssignment
    cout<<a<<endl;

    a %=3; //modulo asssignment
    cout<<a<<endl;
}
```

Output:

PS E:\C++ PROGRAMMES> g++ main.cpp

PS E:\C++ PROGRAMMES> ./a.exe

5

8

5

15

5

2

Home work :

1.Which data types can we apply bitwise operators?

Bitwise operators can be applied to integer data types like int, long, short, char and byte. These operations performed on individual bits and the result is also an integer.

```
#include <iostream>
using namespace std;

int main() {

    int a=5,b=10;
    cout<<(a&b)<<endl;

    char ch='A',bh = 'B';
    cout<<(ch & bh)<<endl;
    cout<<(ch | bh)<<endl;
    cout<<(ch ^ bh)<<endl;
    cout<<(~ch)<<endl;
    cout<<(ch>>1);
    cout<<(ch<<1);
}
```

Output:

PS E:\C++ PROGRAMMES> g++ main.cpp

PS E:\C++ PROGRAMMES> ./a.exe

0

64

67

3

-66

32130

2. Does % operator applied on float data type

We can not apply % on floating values