Operators in C++

Operators are special type of functions, that takes one or more arguments and produces a new value. For example : addition (+), substraction (-), multiplication (*) etc, are all operators. Operators are used to perform various operations on variables and constants.

Types of operators

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Bitwise Operators
- Other Operators

These operators are divided in three parts

Unary Operator - These operators operate or work with a single operand. For example: Increment(++) and Decrement(–) Operators.

E.g. - ++, --, ~

Binary Operator - These operators operate or work with two operands. For example: Addition(+), Subtraction(-), etc.

E.g. - +, -, *, /, %, ==, <, <=, >, >=, !=, &&, ||, !, =, +=, -=, *=, /=, %=, &, |, ^, <<, >>

Ternary Operator - This operator takes three operands, therefore it is known as a Ternary Operator.

E.g. - ? :

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

These are operators used to perform basic mathematical operations. Addition (+) , subtraction (-) , diversion (/) multiplication (*) and modulus (%) are the basic mathematical operators. Modulus operator cannot be used with floating-point numbers.

Example:

```
#include <iostream>
using namespace std;
int main(void)
{
   int num1=5, num2=6, res=0;

   res = num1+num2; // Addition
   cout << "Addition: " << num1 << " + " << num2 << " is: " << res;

   res = num1-num2; // Subtraction
   cout << "\nSubtraction: " << num1 << " - " << num2 << " is: " << res;

   res = num1*num2; // Multiplication
   cout << "\nMultiplication: " << num1 << " * " << num2 << " is: " << res;

   res = num1/num2; // Division
   cout << "\nDivision: " << num1 << " / " << num2 << " is: " << res;

   res = num1%num2; // Modulas
   cout << "\nModulas: " << num1 << " % " << num2 << " is: " << res;
}
```

2. Relational Operators – (==, <, <=, >, >=, !=)

These operators establish a relationship between operands. The relational operators are : less than (<) , grater thatn (>) , less than or equal to (<=), greater than equal to (>=), equivalent (==) and not equivalent (!=).

You must notice that assignment operator is (=) and there is a relational operator, for equivalent (==). These two are different from each other, the assignment operator assigns the value to any Variables, whereas equivalent operator is used to compare values,

Note: Relational operational always result in integer 0(False) or 1(True).

Example:

```
#include <iostream>
using namespace std;
int main(void)
{
   int num1=5, num2=6, res=0;

   res = num1==num2;
   cout << num1 << " == " << num2 << " is: " << res << endl;

   res = num1<num2;
   cout << num1 << " < " << num2 << " is: " << res << endl;

   res = num1<=num2;
   cout << num1 << " <= " << num2 << " is: " << res << endl;

   res = num1>num2;
   cout << num1 << " > " << num2 << " is: " << res << endl;

   res = num1>num2;
   cout << num1 << " >= " << num2 << " is: " << res << endl;

   res = num1!=num2;
   cout << num1 << " != " << num2 << " is: " << res << endl;
}
```

3. Logical Operators – (&&, ||, !)

The logical operators are AND (&&) and OR (||). They are used to combine two different expressions together.

If two statement are connected using AND operator, the validity of both statements will be considered, but if they are connected using OR operator, then either one of them must be valid. These operators are mostly used in loops (especially while loop) and in Decision making.

Example:

```
#include <iostream>
using namespace std;
int main(void)
{
   int num1=5, num2=6, res;

   res = (num1 == num2) && (num1 > num2);
   cout << num1 << "==" << num2 << " && " << num1 << ">" << num2 << "is: 
"<< res << endl;

   res = (num1 == num2) || (num1 < num2);
   cout << num1 << "==" << num2 << " || " << num1 << "<" << num2 << "is: "<< 
res << endl;

   res = !(num1 == num2);
   cout << "!" << num1 << "==" << num2 << "is: "<< res << endl;
}
```

4. Assignment Operators – (=, +=, -=, *=, /=, %=)

Operates '=' is used for assignment, it takes the right-hand side (called rvalue) and copy it into the left-hand side (called lvalue). Assignment operator is the only operator which can be overloaded but cannot be inherited.

Example:

```
#include <iostream>
using namespace std;
int main(void)
{
   Int res=0;

   res = 50;
   cout << "res = 50: " << res << endl;

   res += 50;
   cout << "res += 50: " << res << endl;

   res -= 50;
   cout << "res -= 50: " << res << endl;

   res *= 2;
   cout << "res *= 2: " << res << endl;

   res /= 2;
   cout << "res /= 2: " << res << endl;

   res %= 50;
   cout << "res %= 50: " << res << endl;
}
```

5. Bitwise Operators – (&, |, ^, ~, <<, >>)

There are used to change individual bits into a number. They work with only integral data types like char, int and long and not with floating point values.

1. Bitwise AND operators (&)

```
int num1 = 12, num2 = 25, res=0;

res = num1 & num2;

cout << num1 << " & " << num2 << " = " << res;
```

Explanation:

Bit AND Operation of 12 and 25

```
   0 0 0 0 1 1 0 0
&  0 0 0 1 1 0 0 1
  ________________
   0 0 0 0 1 0 0 0  = 8 (In decimal)
```

2. Bitwise OR operator (|)

```
int num1 = 12, num2 = 25, res=0;

res = num1 | num2;

cout << num1 << " | " << num2 << " = " << res;
```

Explanation:

Bitwise OR Operation of 12 and 25

```
  0 0 0 0 1 1 0 0
| 0 0 0 1 1 0 0 1
 ________________
 0 0 0 1 1 1 0 1  = 29 (In decimal)
```

3. And bitwise XOR operator (^)

```
int num1 = 12, num2 = 25, res=0;

res = num1 ^ num2;

cout << num1 << " ^ " << num2 << " = " << res;
```

Explanation:

Bitwise XOR Operation of 12 and 25

```
  0 0 0 0 1 1 0 0
^ 0 0 0 1 1 0 0 1
 _______________
 0 0 0 1 0 1 0 1  = 21 (In decimal)
```

4. And, bitwise NOT operator (~)

```
int num1 = 12, res=0;

res = ~num1;

cout << " ~ " << num1 << " = " << res;
```

5. Left Shift Operator (<<)

```
int num1 = 12, num2 = 2, res=0;

res = num1 << num2;

cout << num1 << " << " << num2 << " = " << res;
```

6. Right Shift Operator (>>)

```
int num1 = 12, num2 = 2, res=0;

res = num1 >> num2;

cout << num1 << " << " << num2 << " = " << res;
```

6. Other Operators – (Ternary, sizeof, comma, dot, ampersand, asterisk, single arrow operator.)

1. Ternary Operator (? :)

The ternary if-else ? : is an operator which has three operands.

int a = 10;

a > 5 ? cout << "true" : cout << "false"

2. sizeof Operator

The sizeof is a unary operator that returns the size of data (constants, variables, array, structure, etc).

3. Comma Operator (,)

This is used to separate variable names and to separate expressions. In case of expressions, the value of last expression is produced and used.

Example :

int a,b,c; // variables declaration using comma operator

a=b++, c++; // a = c++ will be done.

4. Dot Operator (.)

Is used for direct member selection via the name of variables of type class, struct, and union. It is also known as the direct member access operator. It is a binary operator that helps us to extract the value or the function associated with the particular object, structure or union.

5. Ampersand (&)

Is used as a reference declarator in addition to being the address operator. The meanings are related but not identical. If you take the address of a reference, it returns the address of its target.

6. Asterisk (*)

Is used to declare a pointer. Pointers allow you to refer directly to values in memory, and allow you to modify elements that would otherwise only be copied.

7. Single Arrow Operator (->)

(->) in C++ also known as Class Member Access Operator is a combination of two different operators that is Minus operator (-) and greater than operator (>). It is used to access the public members of a class, structure, or members of union with the help of a pointer variable.

Precedence and Associativity of Operators

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others. For example, the multiplication operator has higher precedence than the addition operator −,

For example, x = 7 + 3 * 2; here x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3 * 2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	**Operator**	**Associativity**
Postfix	expression++ expression--	Left to right
Unary	++expression –-expression +expression –expression ~ !	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< > <= >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= ^= |= <<= >>= >>>=	Right to left

Postfix and Prefix

```
x = 5;                      x = 5;
y = x++;                    y = ++x;
cout << x << “ ” << y;      cout << x << “ ” << y;
Output = 6, 5               Output = 6, 6
```