Programming Language-II

Lecture # 4

Lecture Content

- Operators
- Arithmetic operators
- Relational operators
- Logical operators
- Conditional operators
- MIS Operators
- Bitwise operators
- Operators precedence

Operators

- An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C++ is rich in built-in operators and provide the following types of operators which are listed in next slide.

Operators(Cont…)

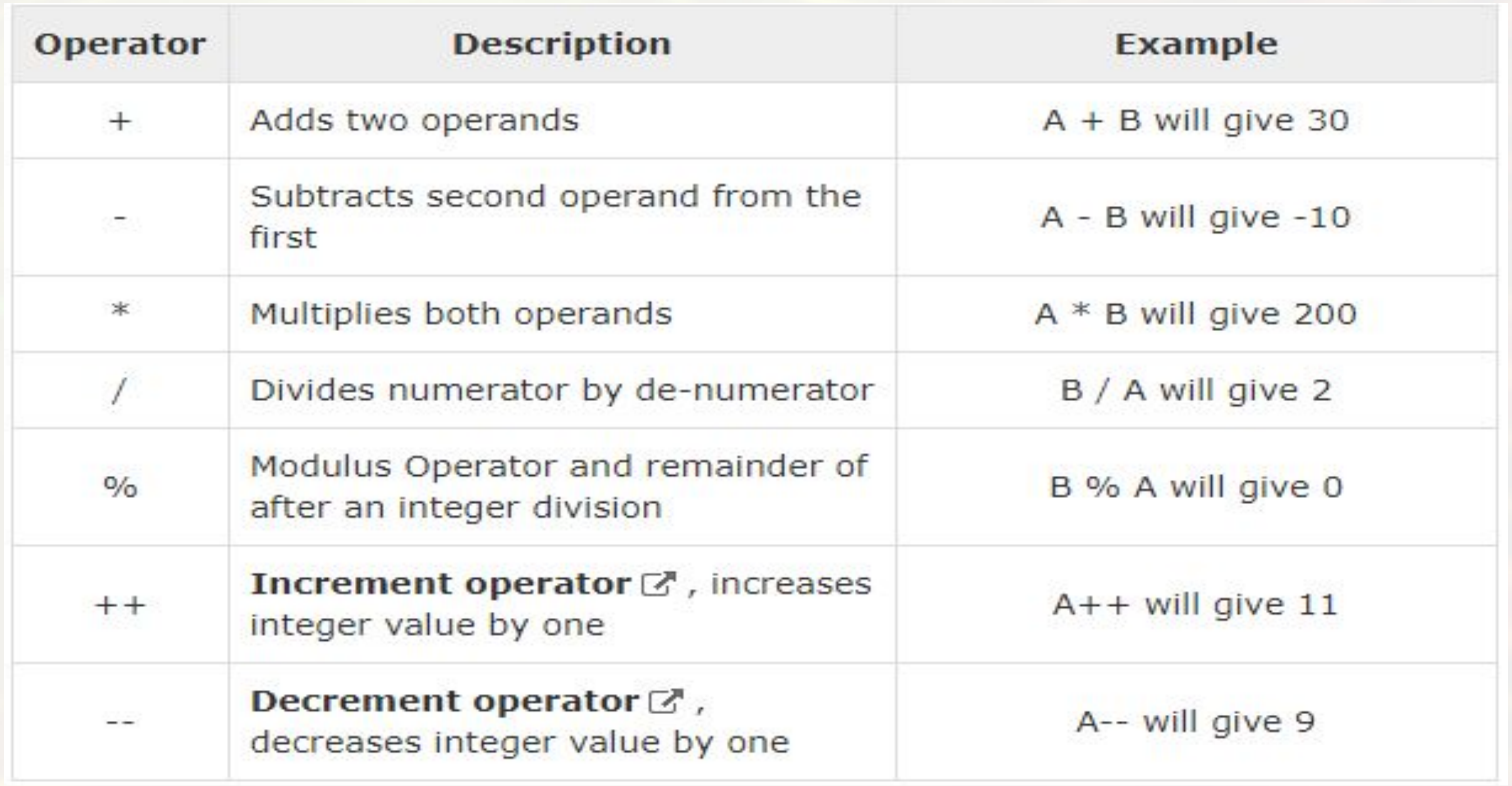
- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Misc Operators
- This lecture will examine the arithmetic, relational, logical, bitwise, assignment and other operators one by one.

Arithmetic operators

- here are following arithmetic operators supported by C++ language
- +(plus)
- -(minus)
- *(multiplication)
- /(division)
- %(modulus)
- ++(increment)
- --(decrement)

Arithmetic operators(Cont…)

- Assume variable A holds 10 and variable B holds 20, then



Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	**Increment operator** , increases integer value by one	A++ will give 11
--	**Decrement operator** , decreases integer value by one	A-- will give 9

Mixed Expressions(Rules)

Two rules apply when evaluating a mixed expression:

When evaluating an operator in a mixed expression:

a. If the operator has the same types of operands (that is, either both integers or both floating-point numbers), the operator is evaluated according to the type of the operands. Integer operands thus yield

an integer result; floating-point numbers yield a floating-point number.

b. If the operator has both types of operands (that is, one is an integer and the other is a floating-point number), then during calculation, the integer is changed to a floating-point number with the decimal

part of zero and the operator is evaluated. The result is a floating point number.

Mixed Expressions(Rules)

2.The entire expression is evaluated according to the precedence rules; the multiplication, division, and modulus operators are evaluated before the addition and subtraction operators. Operators having the same level of precedence are evaluated from left to right. Grouping is allowed for clarity.

Arithmetic operators(Example)

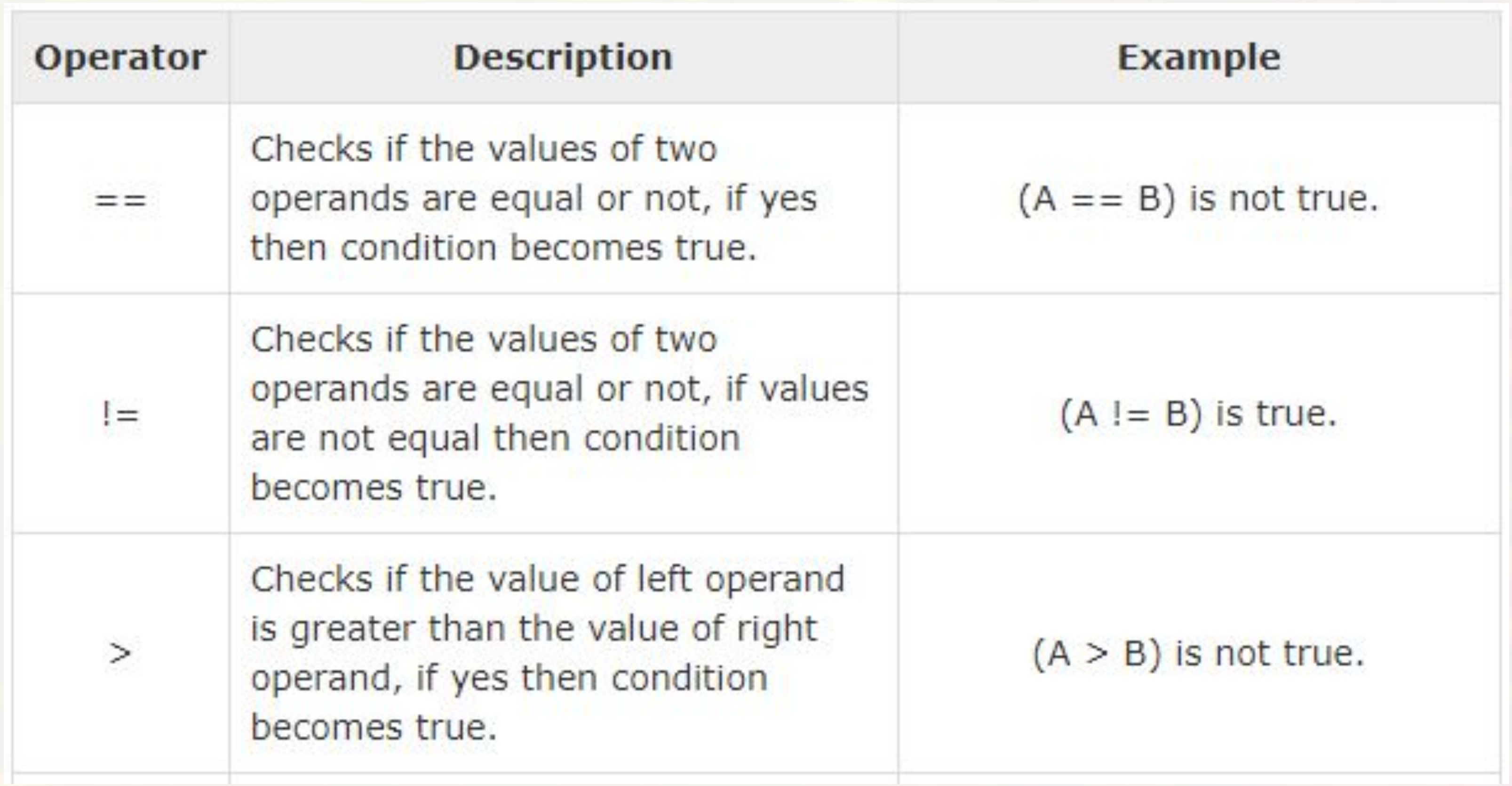
```
#include <iostream>
using namespace std;
int main()
    {
    cout << "3 / 2 + 5.5 = " << 3 / 2 + 5.5 << endl;
    cout << "15.6 / 2 + 5 = " << 15.6 / 2 + 5 << endl;
    cout << "4 + 5 / 2.0 = " << 4 + 5 / 2.0 << endl;
    cout << "4 * 3 + 7 / 5 - 25.5 = "
    << 4 * 3 + 7 / 5 - 25.5
    << endl;
    return 0;
    }
```

Relational operators

- There are following relational operators supported by C++ language
- ==(equal equal)
- !=(not equal)
- <=(less then equal)
- >=(greater then equal)
- <(less then)
- >(greater then)

Relational operators(Cont…)

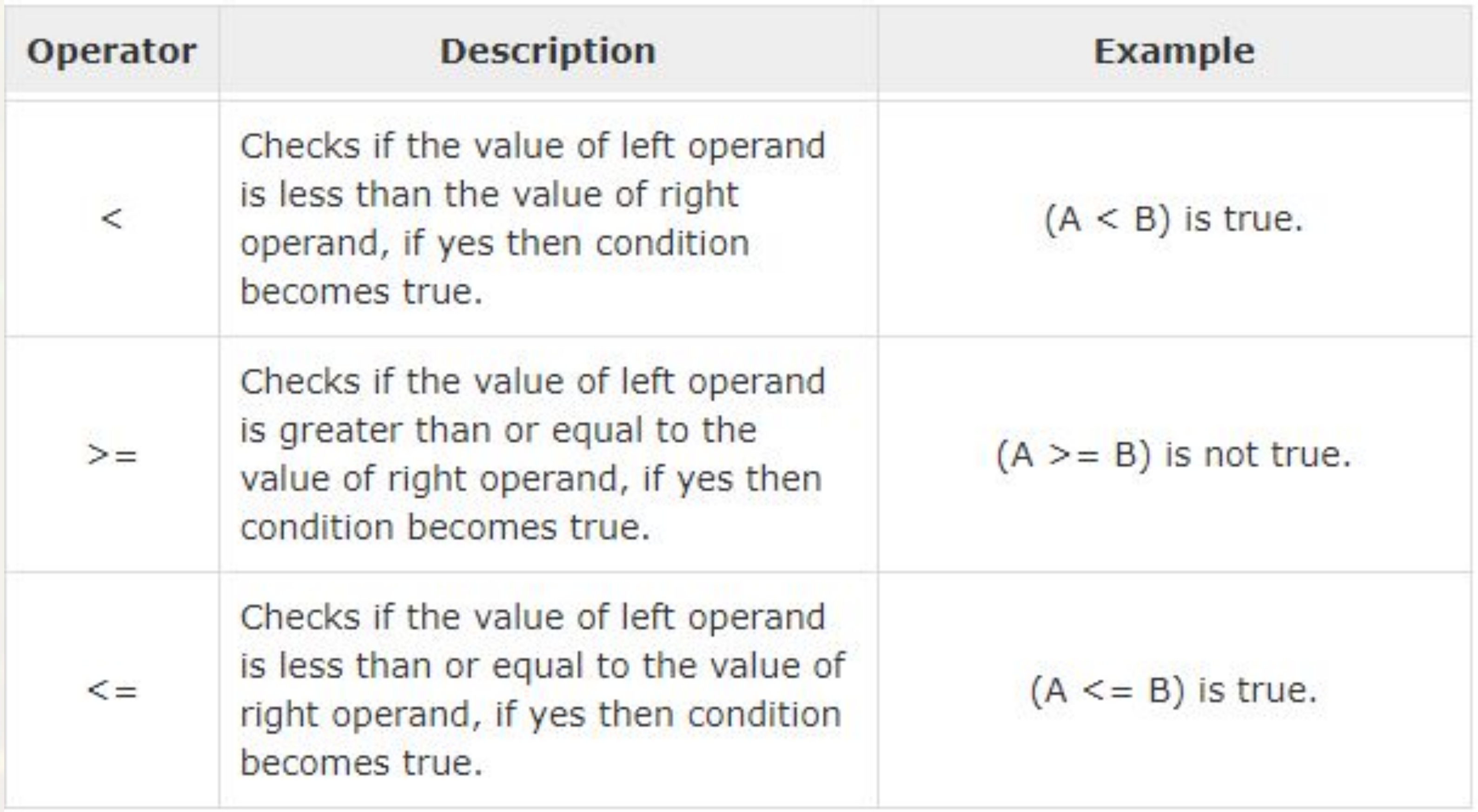
- Assume variable A holds 10 and variable B holds 20, then



Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.

Relational operators(Cont…)

- variable A holds 10 and variable B holds 20, then



Operator	Description	Example
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

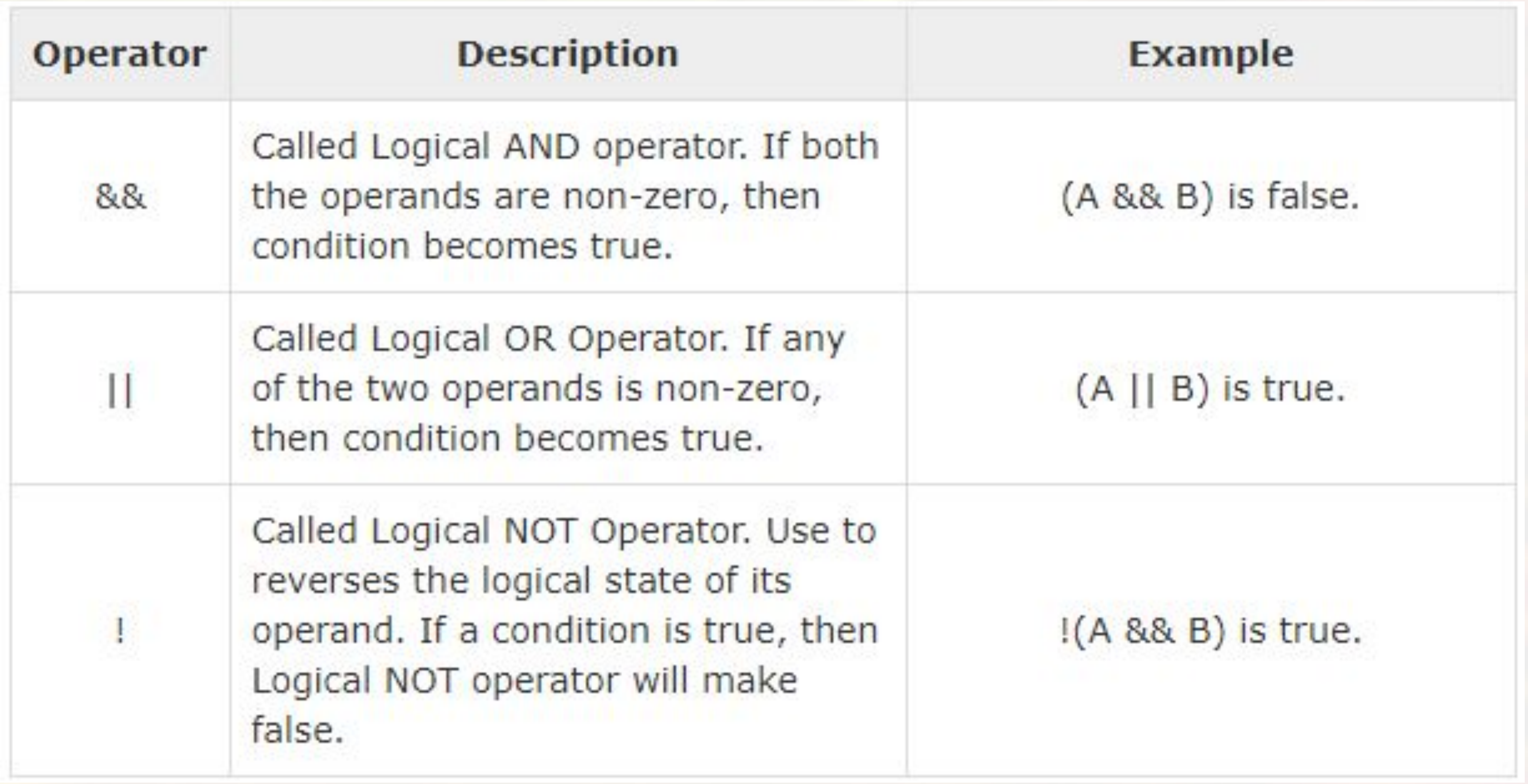
Logical operators

- There are following logical operators supported by C++ language.

- &&(AND)
- ||(OR)
- !(NOT)

Logical operators

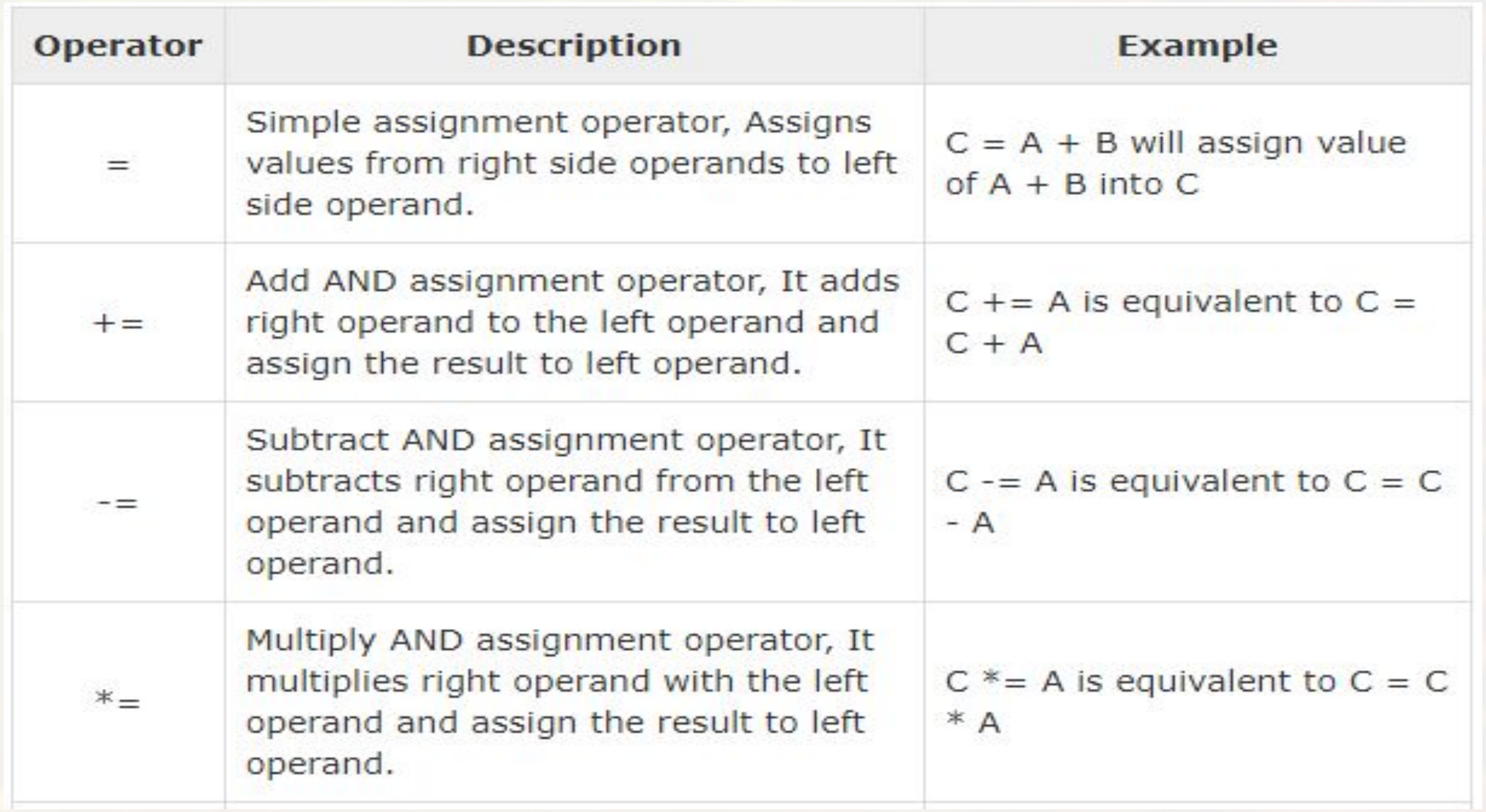
- Assume variable A holds 1 and variable B holds 0, then



Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	!(A && B) is true.

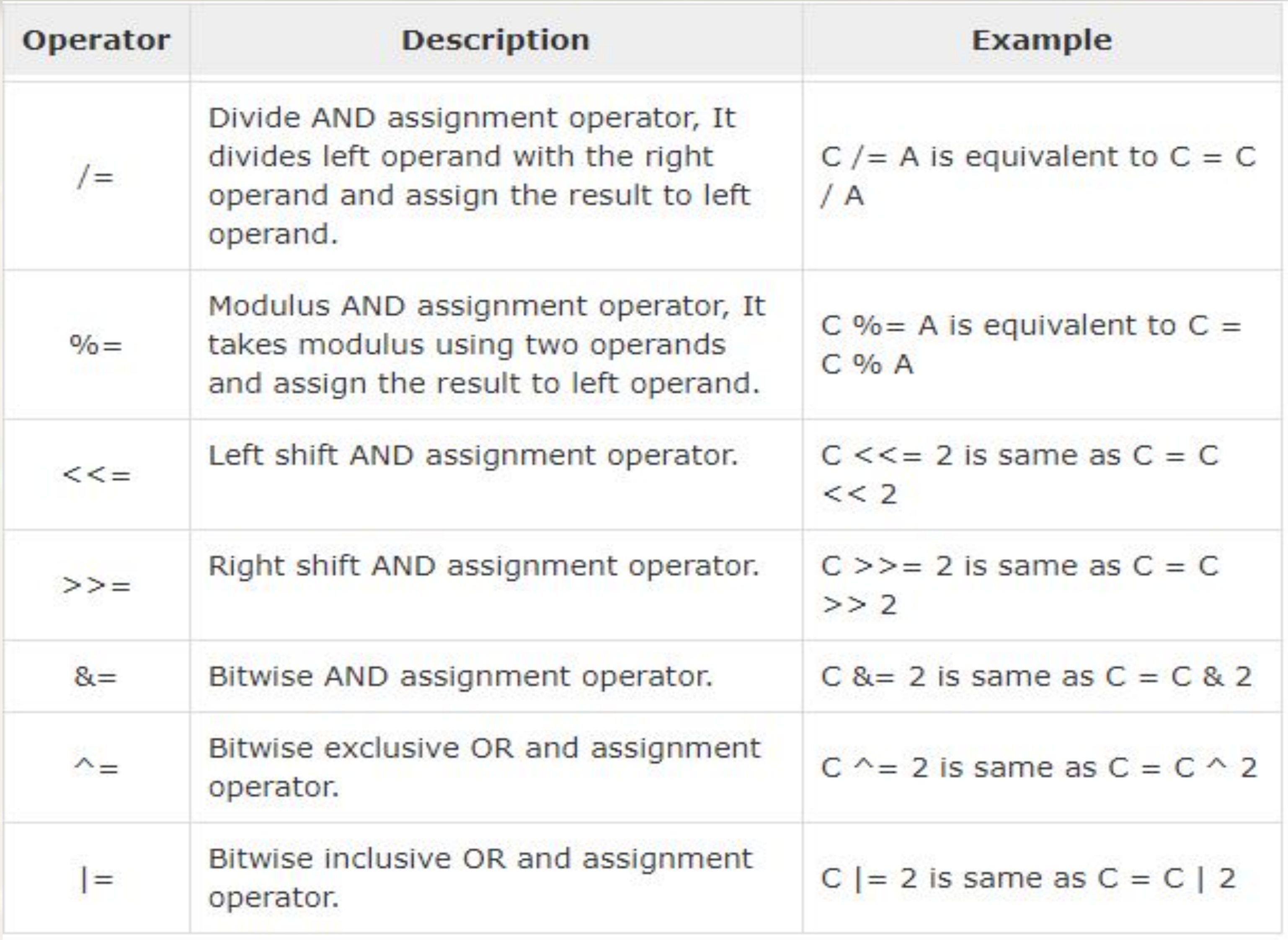
Assignment Operators

- There are following assignment operators supported by C++ language



Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A

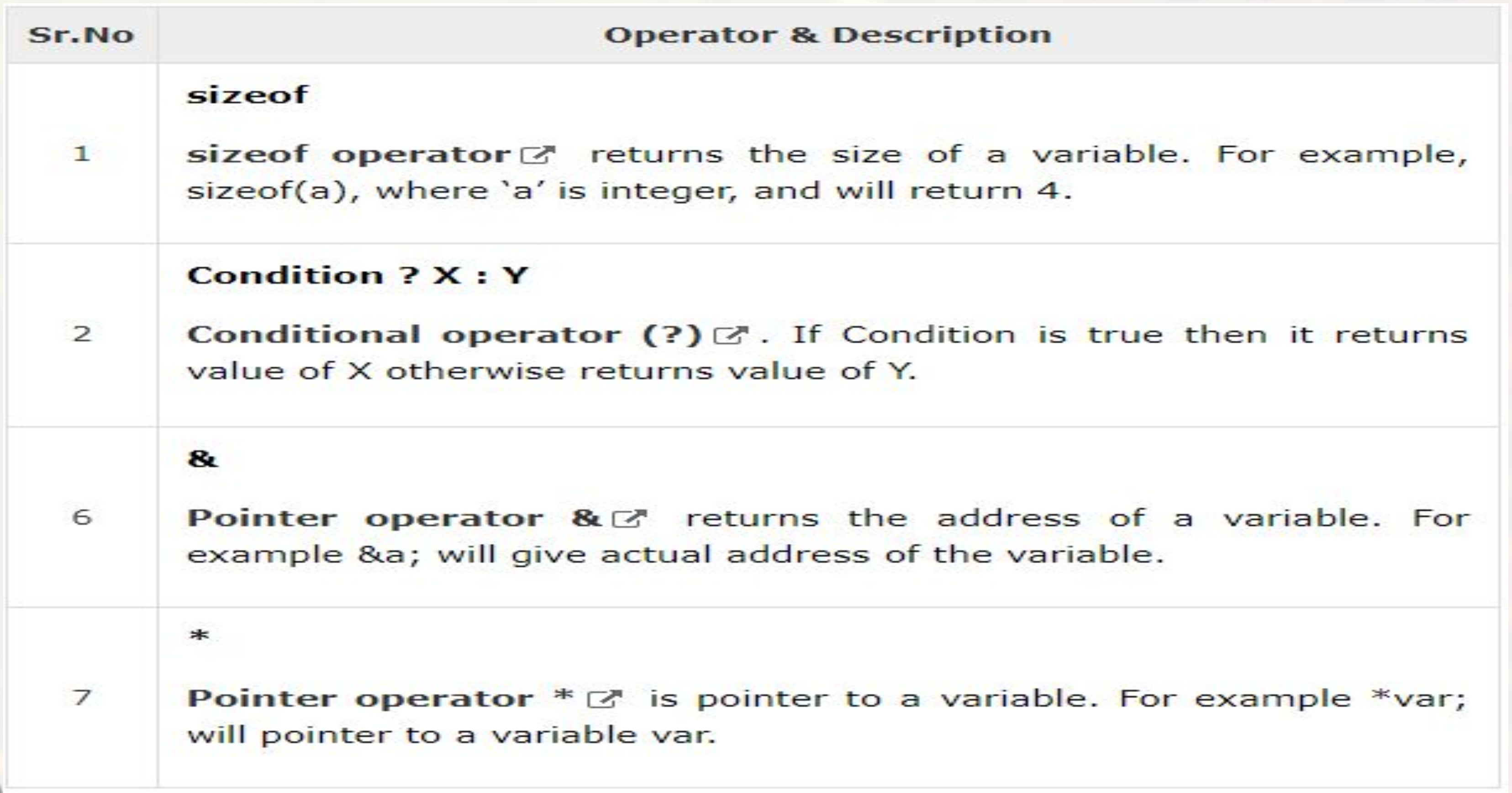
Assignment Operators



Operator	Description	Example
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

MIX Operators

- The following table lists some other operators that C++ supports.



Sr.No	Operator & Description
1	**sizeof** **sizeof operator** returns the size of a variable. For example, sizeof(a), where 'a' is integer, and will return 4.
2	**Condition ? X : Y** **Conditional operator (?)**. If Condition is true then it returns value of X otherwise returns value of Y.
6	**&** **Pointer operator &** returns the address of a variable. For example &a; will give actual address of the variable.
7	***** **Pointer operator *** is pointer to a variable. For example *var; will pointer to a variable var.

Ternary operator example:

```
#include <iostream>
using namespace std;
int main()
{
        int a = 5;
        int b = 10, larger;
        larger = ((a > b) ? a: b);
        cout << larger << endl;

        system("pause");
        return 0;
}
```

Operators Precedence

- 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.

Bitwise operators

- Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ are as follows.

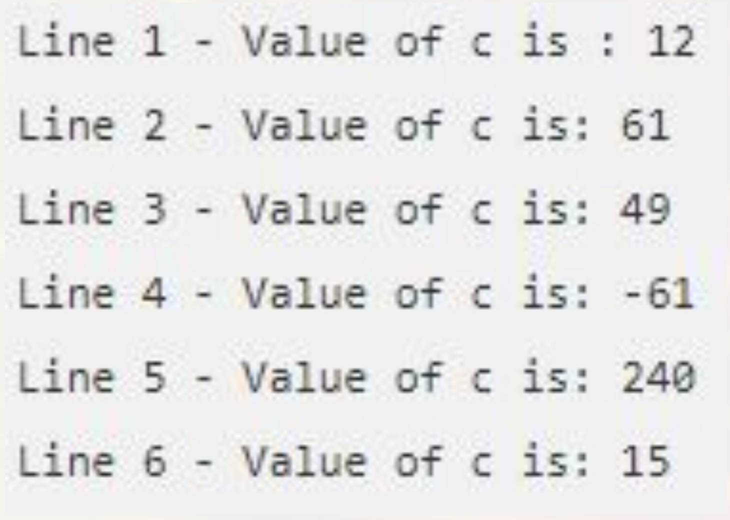
- Assume if A = 60; and B = 13; now in binary format they will be as follows −

```
A = 0011 1100
B = 0000 1101
-----------------
A&B = 0000 1100
A|B = 0011 1101
A^B = 0011 0001
~A  = 1100 0011
```

Example

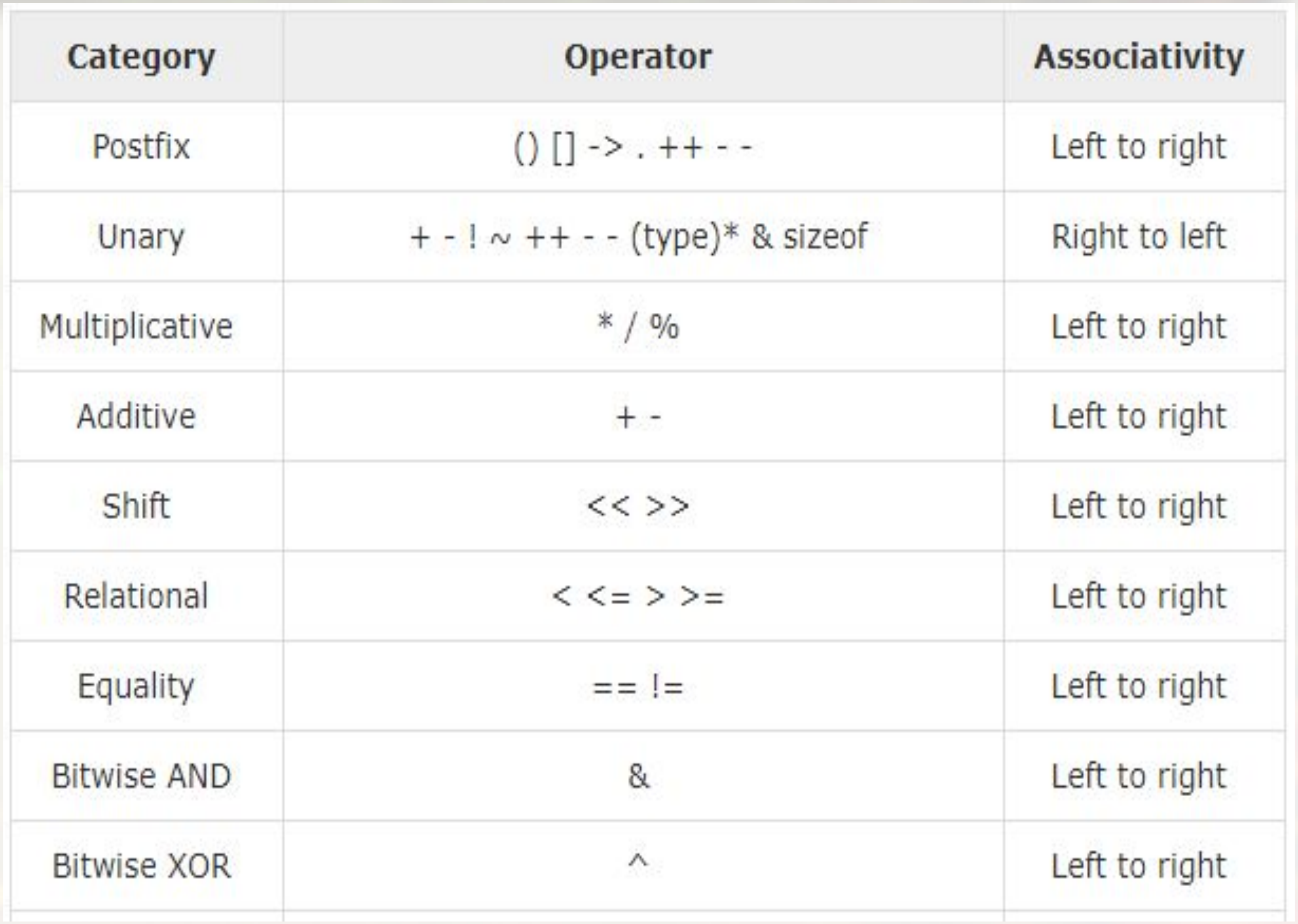
```
#include <iostream>
using namespace std;
int main() {
        unsigned int a = 60; // 60 = 0011 1100

        unsigned int b = 13; // 13 = 0000 1101
        int c = 0;
        c = a & b; // 12 = 0000 1100
        cout << "Line 1 - Value of c is : " << c << endl ;
        c = a | b; // 61 = 0011 1101
        cout << "Line 2 - Value of c is: " << c << endl ;
        c = a ^ b; // 49 = 0011 0001
        cout << "Line 3 - Value of c is: " << c << endl ;
        c = ~a; // -61 = 1100 0011
        cout << "Line 4 - Value of c is: " << c << endl ;
        c = a << 2; // 240 = 1111 0000
        cout << "Line 5 - Value of c is: " << c << endl ;
        c = a >> 2; // 15 = 0000 1111
        cout << "Line 6 - Value of c is: " << c << endl ;
        return 0;
}
```

Output



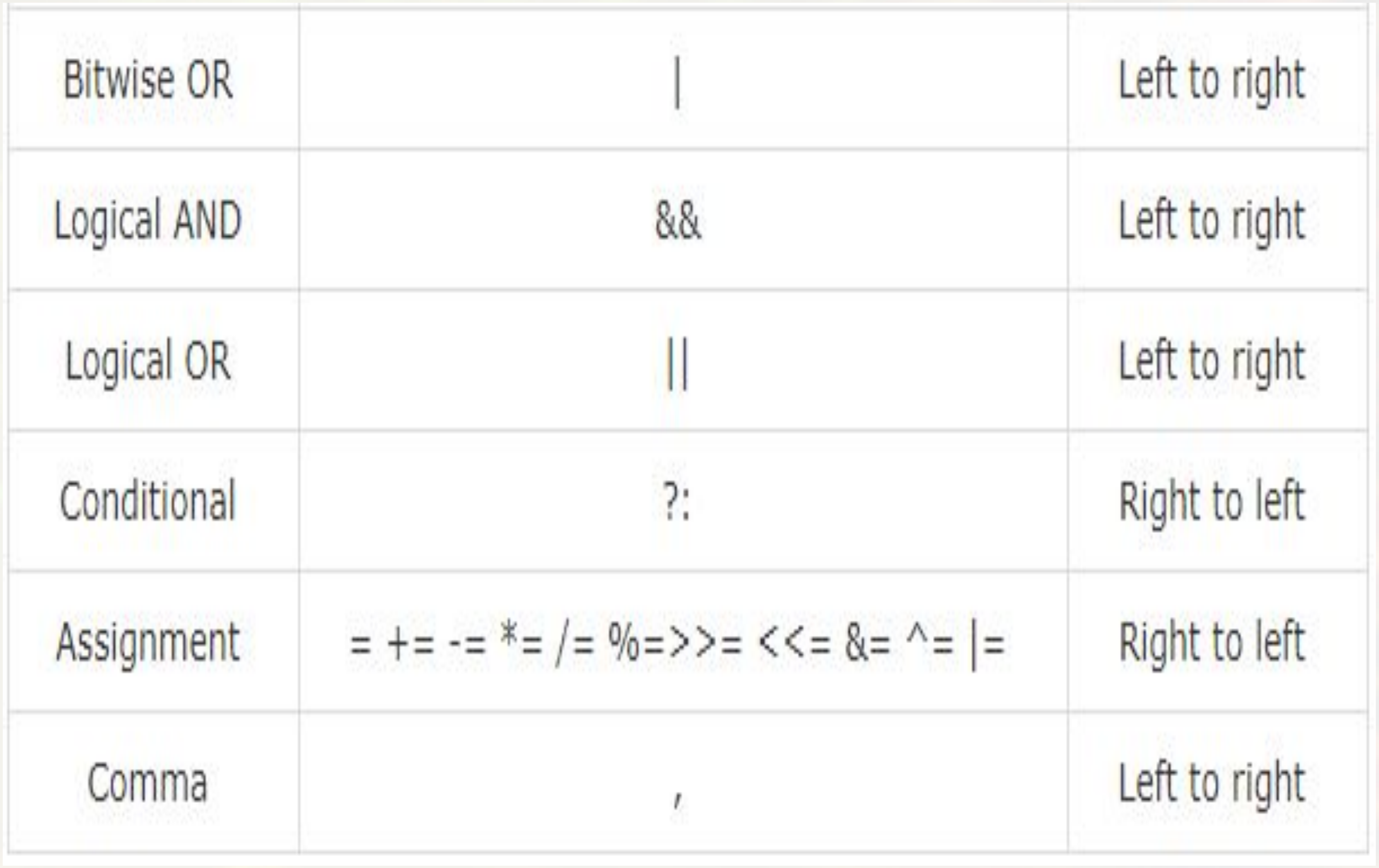
```
Line 1 - Value of c is : 12
Line 2 - Value of c is: 61
Line 3 - Value of c is: 49
Line 4 - Value of c is: -61
Line 5 - Value of c is: 240
Line 6 - Value of c is: 15
```

Operators Precedence(Cont…)



Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	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

Operators Precedence(Cont…)



Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right