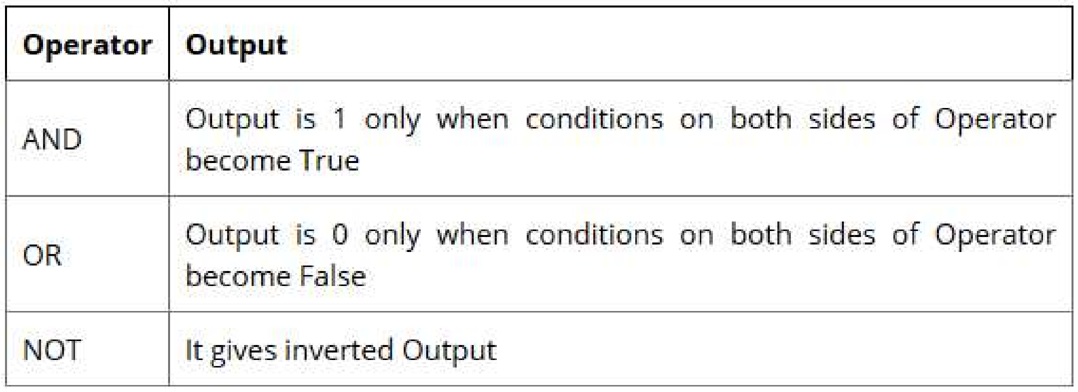
Operators	Name of the Operator	Type
&&	AND Operator	Binary
||	OR Operator	Binary
!	NOT Operator	Unary

Logical Operators

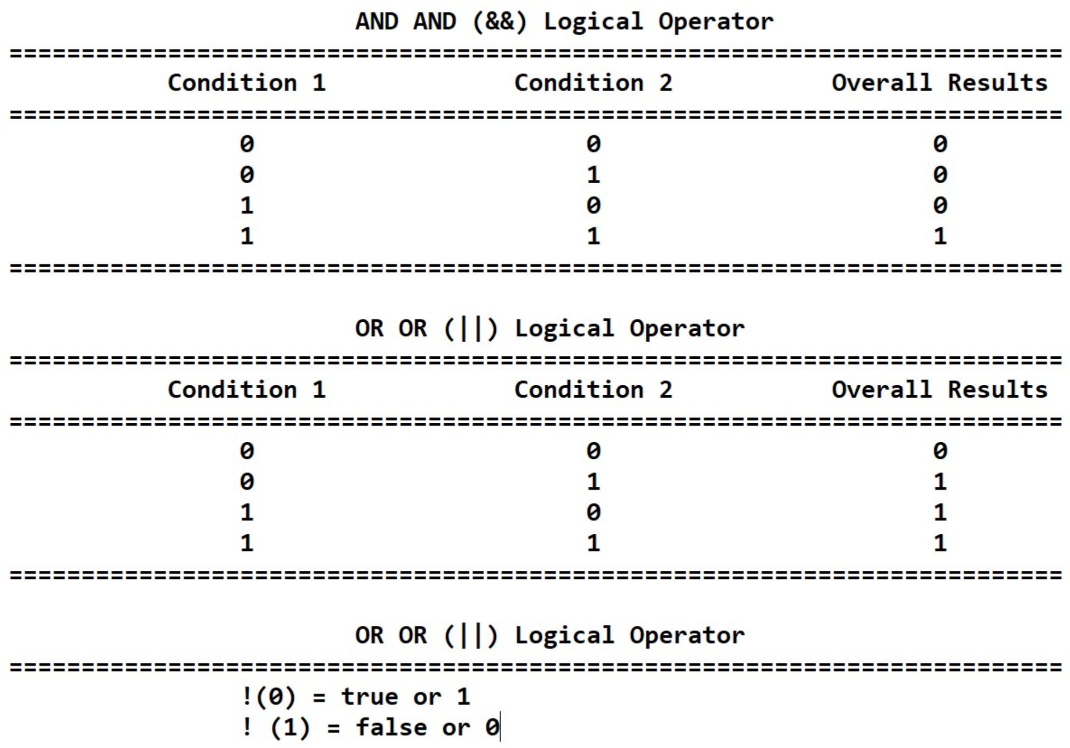
LOGICAL OPERATORS ARE USED IF WE WANT TO COMPARE MORE THAN ONE CONDITION.

Logical Operators



Operator	Output
AND	Output is 1 only when conditions on both sides of Operator become True
OR	Output is 0 only when conditions on both sides of Operator become False
NOT	It gives inverted Output

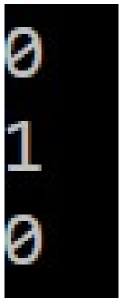
Logical Operators



```
                        AND AND (&&) Logical Operator
=========================================================================
           Condition 1             Condition 2           Overall Results
=========================================================================
                0                       0                       0
                0                       1                       0
                1                       0                       0
                1                       1                       1
=========================================================================

                        OR OR (||) Logical Operator
=========================================================================
           Condition 1             Condition 2           Overall Results
=========================================================================
                0                       0                       0
                0                       1                       1
                1                       0                       1
                1                       1                       1
=========================================================================

                        OR OR (||) Logical Operator
=========================================================================
                !(0) = true or 1
                ! (1) = false or 0
```

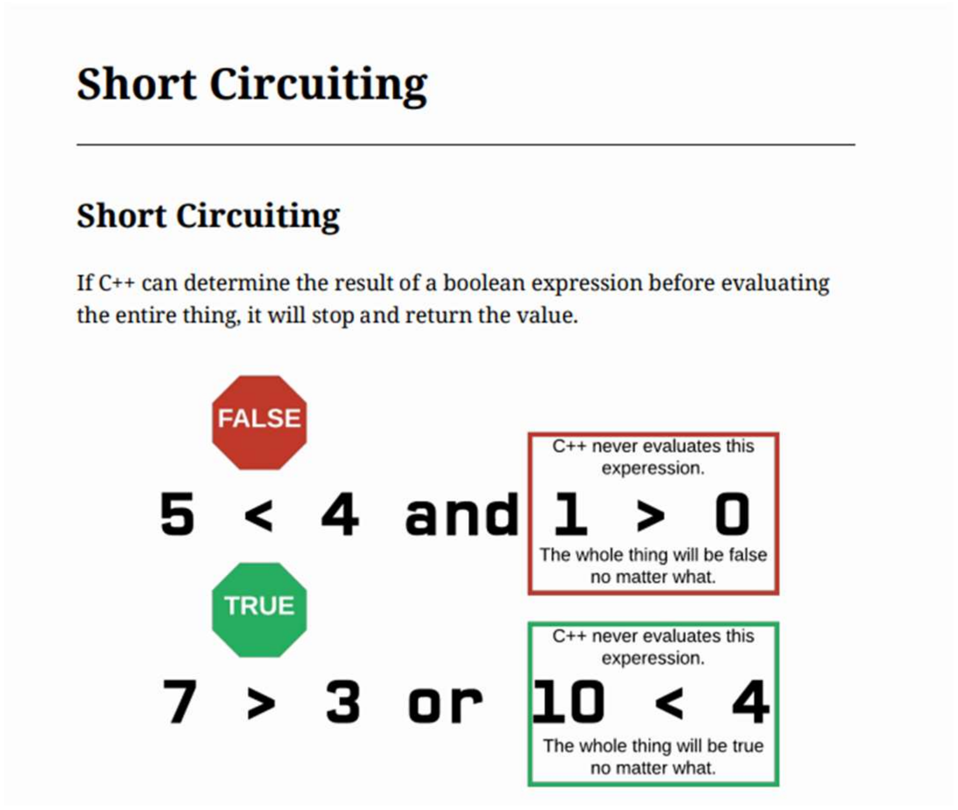
Logical Operators

```
cout <<((10 >= 20) && (10 == 10))<<endl ;
cout <<((10 >= 20) || (10 == 10))<<endl ;
cout <<(!(10 <= 20) || !(10 == 10))<<endl ;
```



```
0
1
0
```

Short Circuiting



Short Circuiting

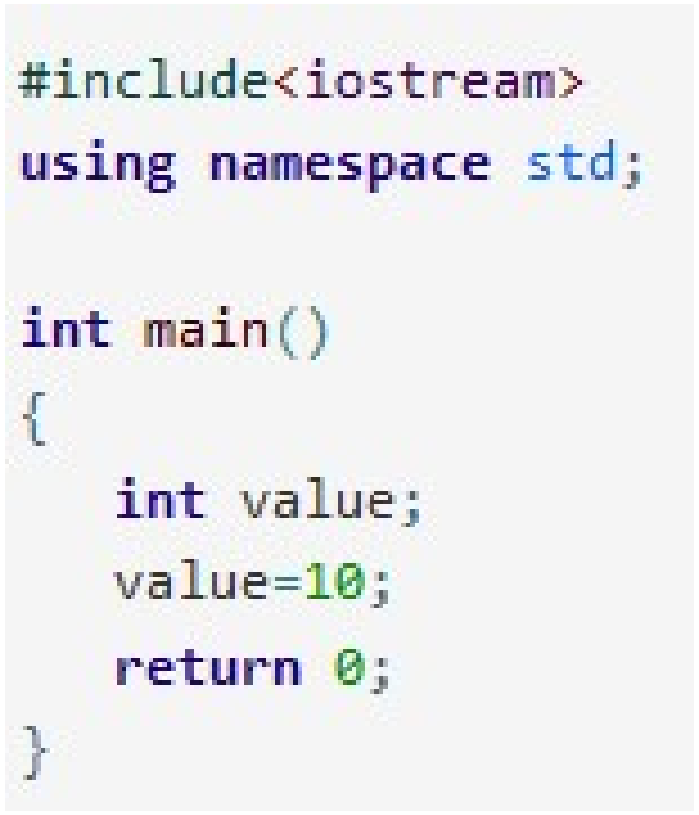
Short Circuiting

If C++ can determine the result of a boolean expression before evaluating the entire thing, it will stop and return the value.

FALSE
5 < 4 and 1 > 0
C++ never evaluates this expression.
The whole thing will be false no matter what.
TRUE
7 > 3 or 10 < 4
C++ never evaluates this experssion.
The whole thing will be true no matter what.

Assignment Operators

- To assign the values to variables
- Assignment Operator is denoted by equal to (=) sign.
- This operator copies the value at the right side of the operator into the left side variable.
- Assignment Operator is binary operator.
- In this example, 10 is assigned to variable named value.

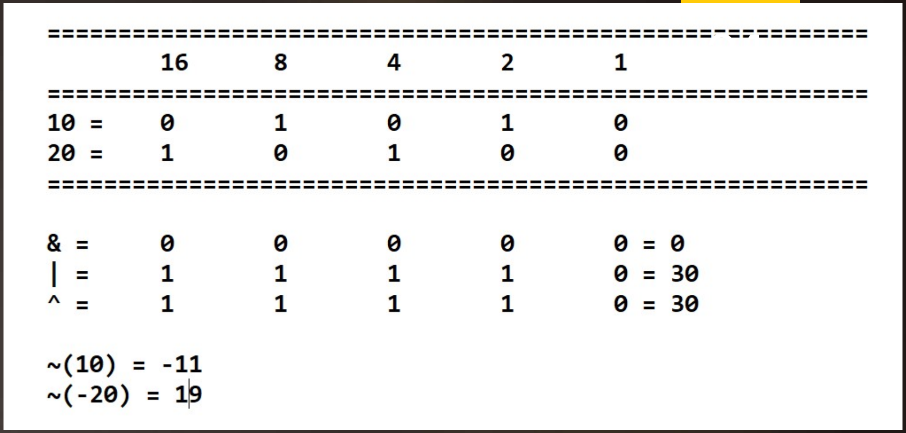


```
#include<iostream>
using namespace std;

int main()
{
    int value;
    value=10;
    return 0;
}
```

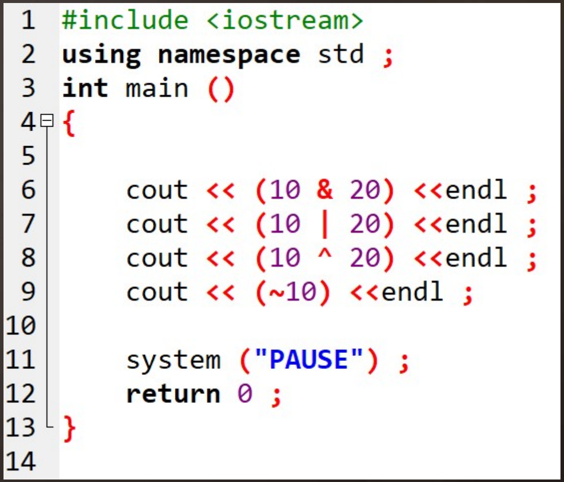
Bitwise Operators

- Operate on the individual data bit.
- C++ Bitwise Operators operate on Integer and character data types only.
- C++ Bitwise Operators do not operate on float, double.
- There are four bitwise operators

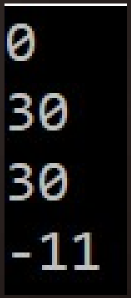
1. Bitwise AND (&)
2. Bitwise OR (|)
3. Bitwise XOR (^)
4. Bitwise One's Compliment (~)



```
=============================================================
        16      8       4       2       1
=============================================================
10 =    0       1       0       1       0
20 =    1       0       1       0       0
=============================================================

& =     0       0       0       0       0 = 0
| =     1       1       1       1       0 = 30
^ =     1       1       1       1       0 = 30

~(10) = -11
~(-20) = 19
```

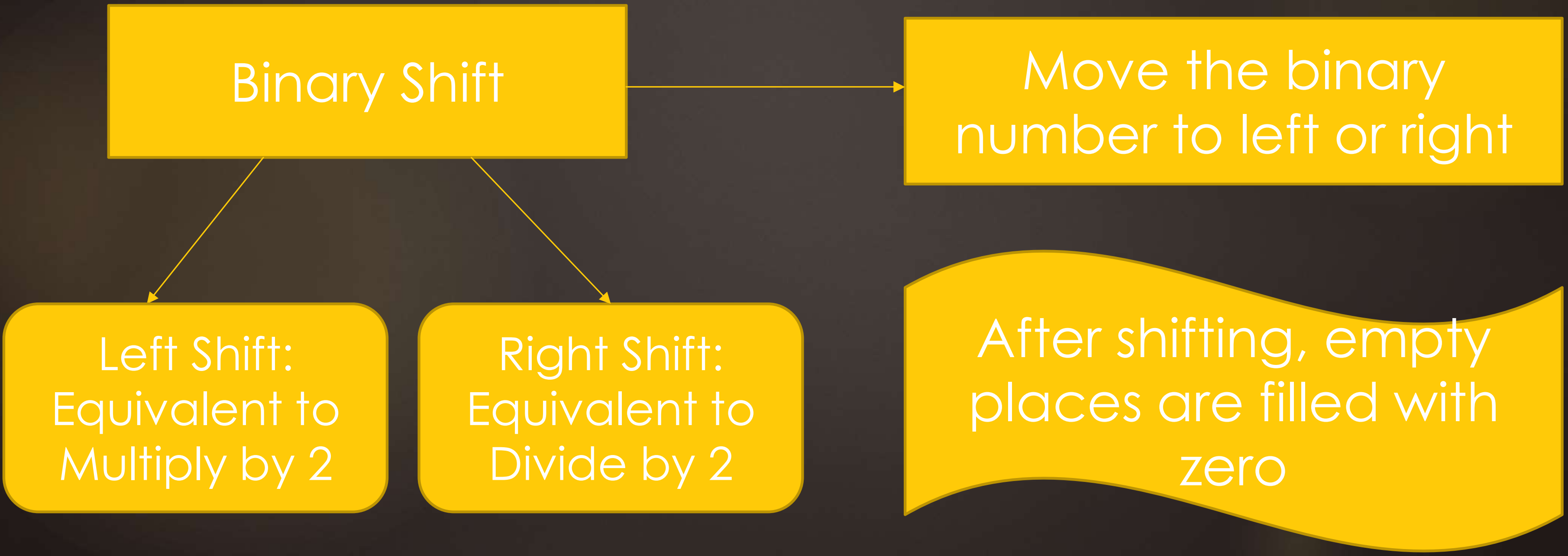


```
#include <iostream>
using namespace std ;
int main ()
{

    cout << (10 & 20) <<endl ;
    cout << (10 | 20) <<endl ;
    cout << (10 ^ 20) <<endl ;
    cout << (~10) <<endl ;

    system ("PAUSE") ;
    return 0 ;
}

```



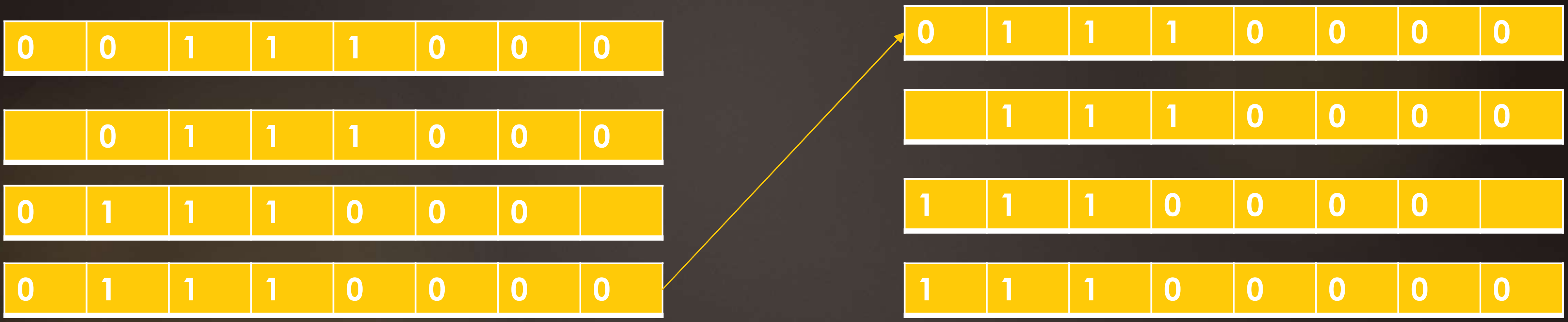
```
0
30
30
-11
```

LOGICAL BINARY SHIFTS


Binary Shift
Move the binary number to left or right
Left Shift: Equivalent to Multiply by 2
Right Shift: Equivalent to Divide by 2
After shifting, empty places are filled with zero

LEFT SHIFT

Perform Left Shifting Two places to the left



First Left Shift Done

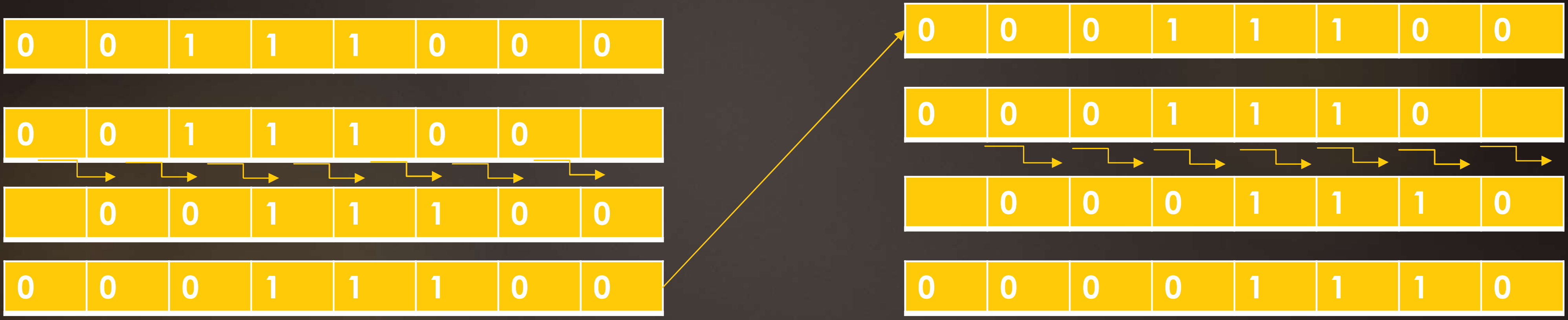
Denary of Original Number is: 56
What should be the denary after first left shift?

Second Left Shift Done

Denary After Second Right Shift will be:
$56*2^{2}=224$

RIGHT SHIFT

Perform RIGHT Shifting Two places to the RIGHT



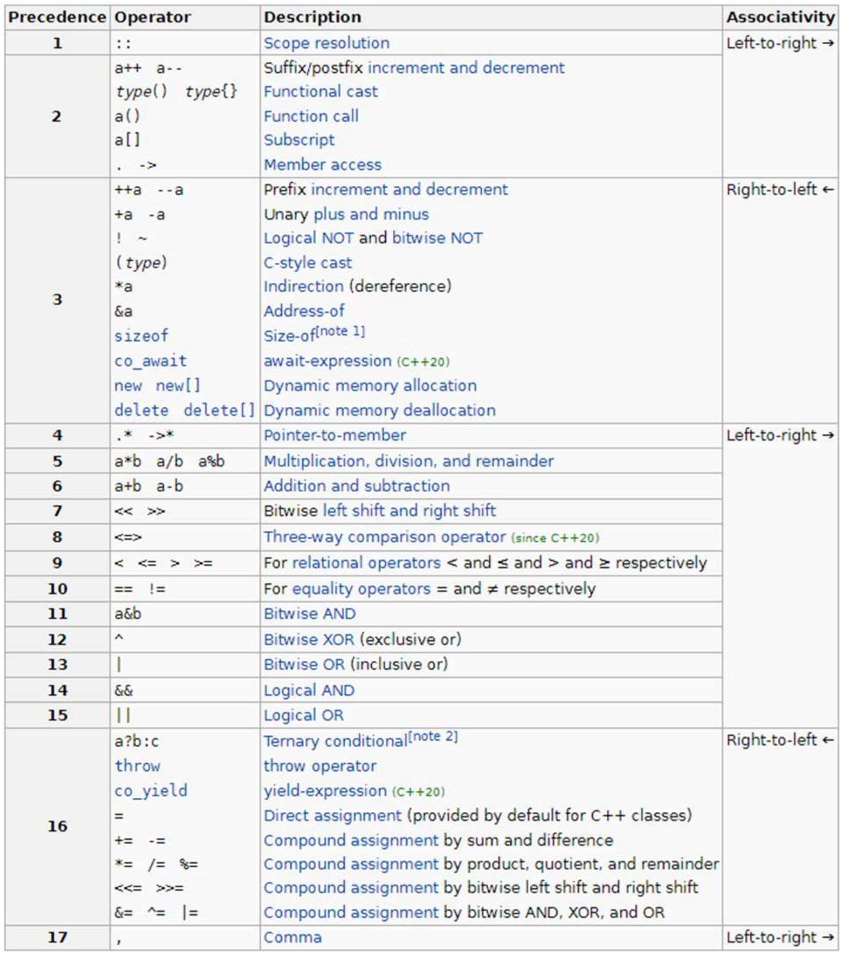
First Right Shift Done

Denary of Original Number is: 56
What should be the denary after first Right shift?

Second Left Shift Done

Denary After Second Right Shift will be:
$56/2^{2}$=14

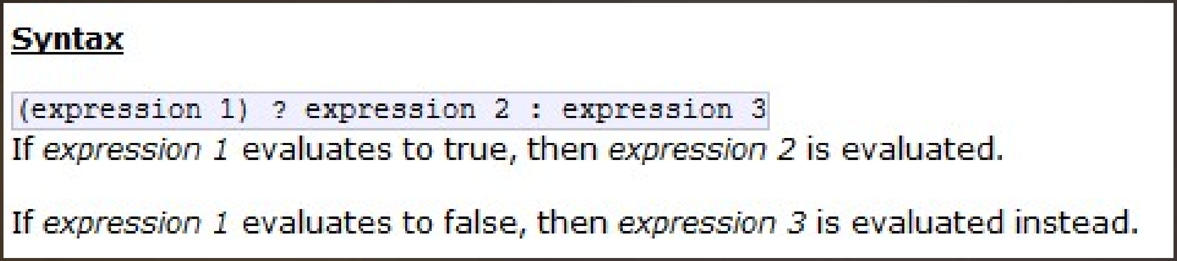
Operator precedence and associativity



Precedence	Operator	Description	Associativity
1	`::`	Scope resolution	Left-to-right →
2	`a++ a--`	Suffix/postfix increment and decrement	
	type`()` *type*`{}`	Functional cast	
	`a()`	Function call	
	`a[]`	Subscript	
	`. ->`	Member access	
3	`++a --a`	Prefix increment and decrement	Right-to-left ←
	`+a -a`	Unary plus and minus	
	`! ~`	Logical NOT and bitwise NOT	
	`(`*type*`)`	C-style cast	
	`*a`	Indirection (dereference)	
	`&a`	Address-of	
	`sizeof`	Size-of[note 1]	
	`co_await`	await-expression (C++20)	
	`new new[]`	Dynamic memory allocation	
	`delete delete[]`	Dynamic memory deallocation	
4	`.* ->*`	Pointer-to-member	Left-to-right →
5	`a*b a/b a%b`	Multiplication, division, and remainder	
6	`a+b a-b`	Addition and subtraction	
7	`<< >>`	Bitwise left shift and right shift	
8	`<=>`	Three-way comparison operator (since C++20)	
9	`< <= > >=`	For relational operators < and ≤ and > and ≥ respectively	
10	`== !=`	For equality operators = and ≠ respectively	
11	`a&b`	Bitwise AND	
12	`^`	Bitwise XOR (exclusive or)	
13	`|`	Bitwise OR (inclusive or)	
14	`&&`	Logical AND	
15	`||`	Logical OR	
16	`a?b:c`	Ternary conditional[note 2]	Right-to-left ←
	`throw`	throw operator	
	`co_yield`	yield-expression (C++20)	
	`=`	Direct assignment (provided by default for C++ classes)	
	`+= -=`	Compound assignment by sum and difference	
	`*= /= %=`	Compound assignment by product, quotient, and remainder	
	`<<= >>=`	Compound assignment by bitwise left shift and right shift	
	`&= ^= |=`	Compound assignment by bitwise AND, XOR, and OR	
17	`,`	Comma	Left-to-right →

Ternary Operators

- The ternary or conditional operator is an operator used in C++.
- Sign is **?:**
- This operator returns one of two values depending on the result of an expression.



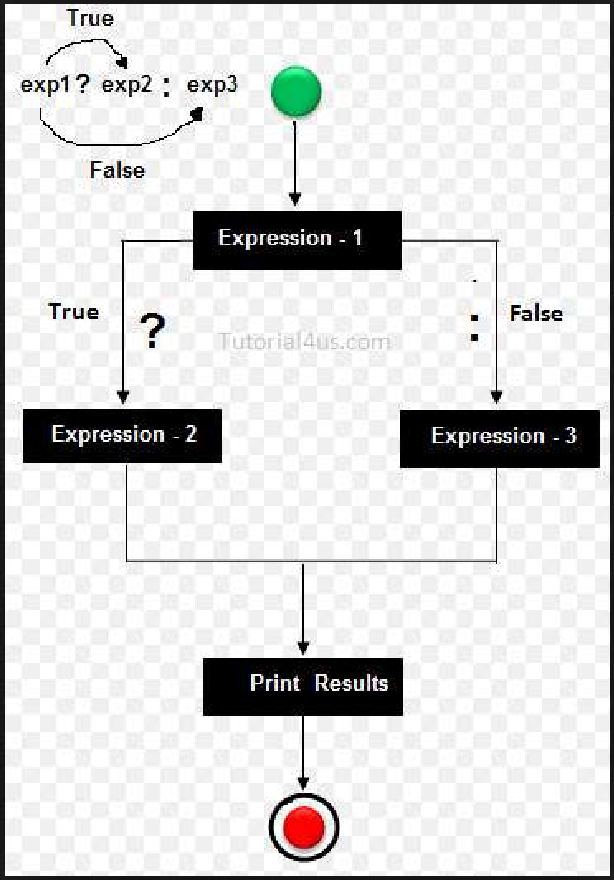
Syntax

```
(expression 1) ? expression 2 : expression 3
```

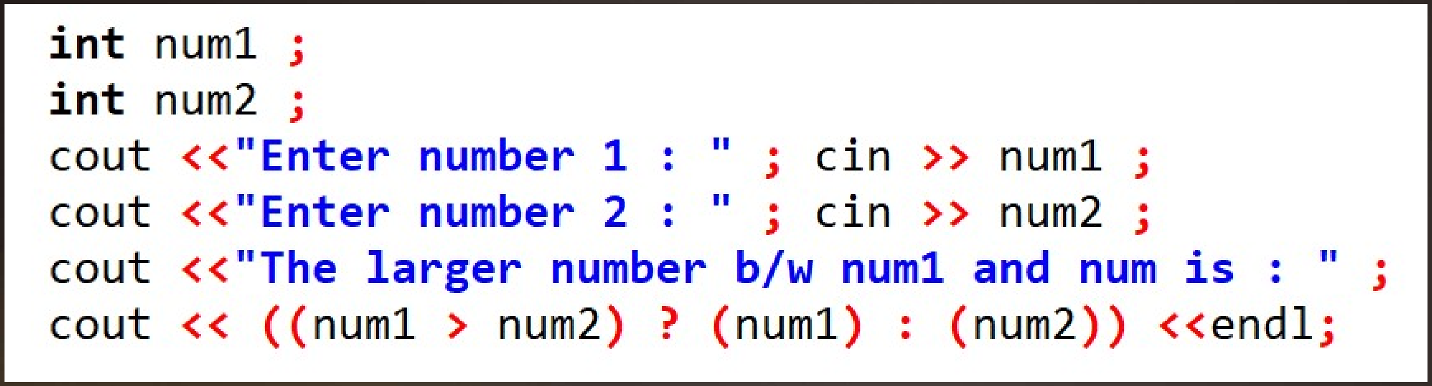
If *expression 1* evaluates to true, then *expression 2* is evaluated.

If *expression 1* evaluates to false, then *expression 3* is evaluated instead.

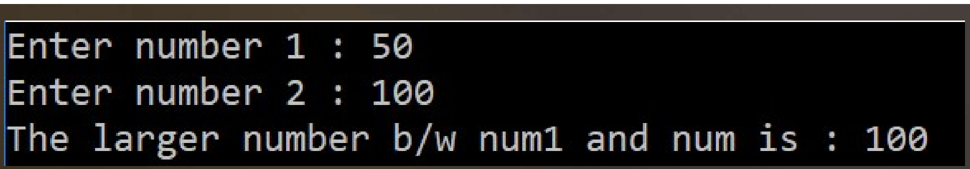
(condition) ? (if_true) : (if_false)



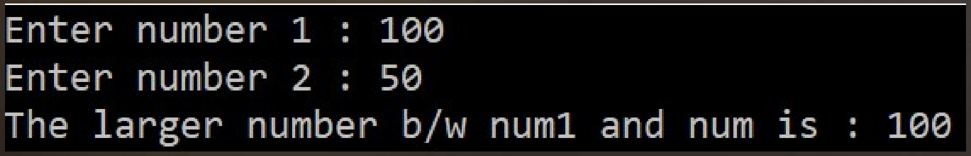
Ternary Operators



```cpp
int num1 ;
int num2 ;
cout <<"Enter number 1 : " ; cin >> num1 ;
cout <<"Enter number 2 : " ; cin >> num2 ;
cout <<"The larger number b/w num1 and num is : " ;
cout << ((num1 > num2) ? (num1) : (num2)) <<endl;
```

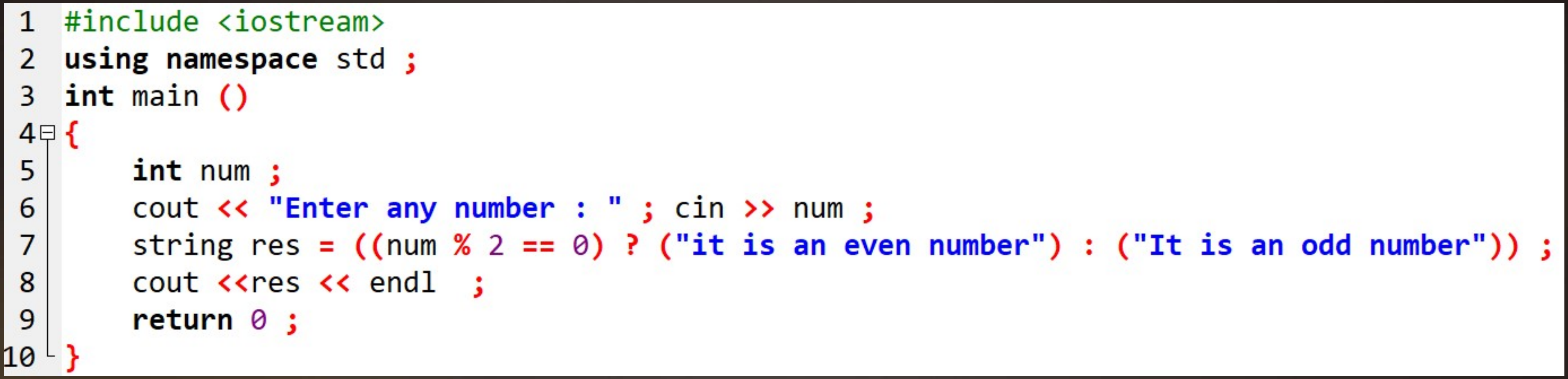


```
Enter number 1 : 50
Enter number 2 : 100
The larger number b/w num1 and num is : 100
```

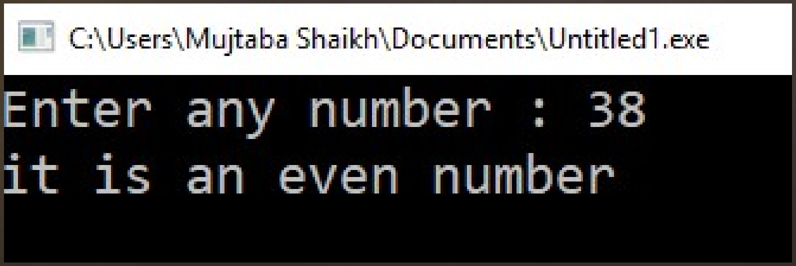


```
Enter number 1 : 100
Enter number 2 : 50
The larger number b/w num1 and num is : 100
```

Ternary Operators



```cpp
#include <iostream>
using namespace std ;
int main ()
{
    int num ;
    cout << "Enter any number : " ; cin >> num ;
    string res = ((num % 2 == 0) ? ("it is an even number") : ("It is an odd number")) ;
    cout <<res << endl  ;
    return 0 ;
}
```



```
Enter any number : 38
it is an even number
```