IOOP- Unit2

INTRODUCTION TO OBJECT ORIENTED PROGRAMMING

Introduction : Operators in C++

- An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.

1) **Assignment Operator**
2) **Arithmetic Operator**
3) **Relational Operator**
4) **Logical Operators**
5) **Shorthand Arithmetic Assignment Operators**
6) **Increment/Decrement Operators**
7) **Conditional Operator**
8) **Sizeof () Operator**
9) **Comma Operator**
10) **Ternary Operator**
11) **Operator precedence levels and Associativity**

Assignment Operators

Simple Assignment Operator	Shorthand Assignment Operator
x = x+1	x += 1
x = x-1	x -= 1
x = x*(n+1)	x *= n+1
x = x / (n+1)	x /= n+1
x = x % (n+1)	x %= n+1

<< Shift Left

SYNTAX	BINARY FORM	VALUE
x = 7;	00000111	7
x=x<<1;	00001110	14
x=x<<3;	01110000	112
x=x<<2;	11000000	192

>> Shift Right

SYNTAX	BINARY FORM	VALUE
x = 192;	11000000	192
x=x>>1;	01100000	96
x=x>>2;	00011000	24
x=x>>3;	00000011	3

Arithmetic Operators

Operator	Meaning	Example	Result
+	Addition	10 + 2	12
-	Subtraction	10 – 2	8
*	Multiplication	10 * 2	20
/	Division	10 / 2	5
%	Modulus (remainder)	10 % 2	0
++	Increment	a++ (consider a = 10)	11
--	Decrement	a-- (consider a = 10)	9
+=	Addition Assignment	a += 10 (consider a = 10)	20
-=	Subtraction assignment	a -= 10 (consider a = 10)	0
*=	Multiplication assignment	a *= 10 (consider a = 10)	100
/=	Division assignment	a /= 10 (consider a = 10)	1
%=	Modulus assignment	a %= 10 (consider a = 10)	0

Relational Operators

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
!=	Not equal to	5!=2	True

Logical Operators

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

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

Show Examples

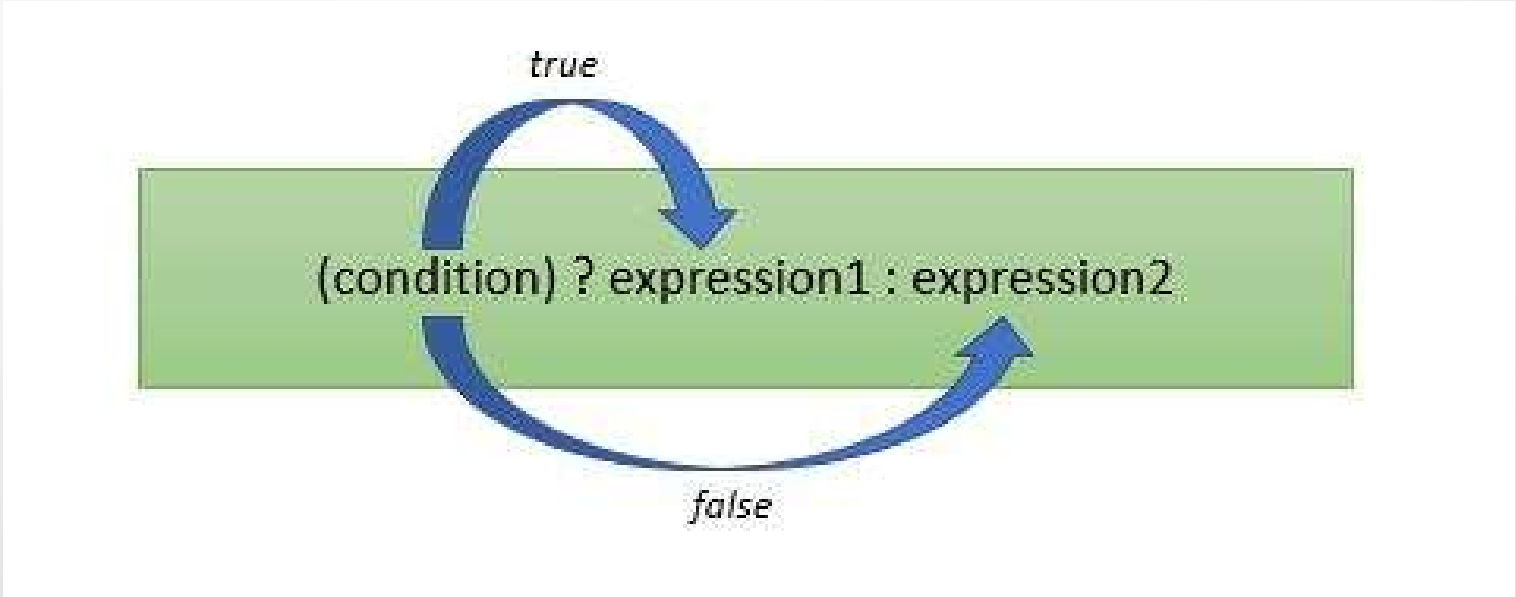
Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses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

Increment Decrement Operator

OPERATOR	MEANING
++a	Increment a by 1, then use new value of a
a++	Use value of a, then increment a by 1
--b	Decrement a by 1, then use new value of a
b--	Use value of a, then decrement a by 1

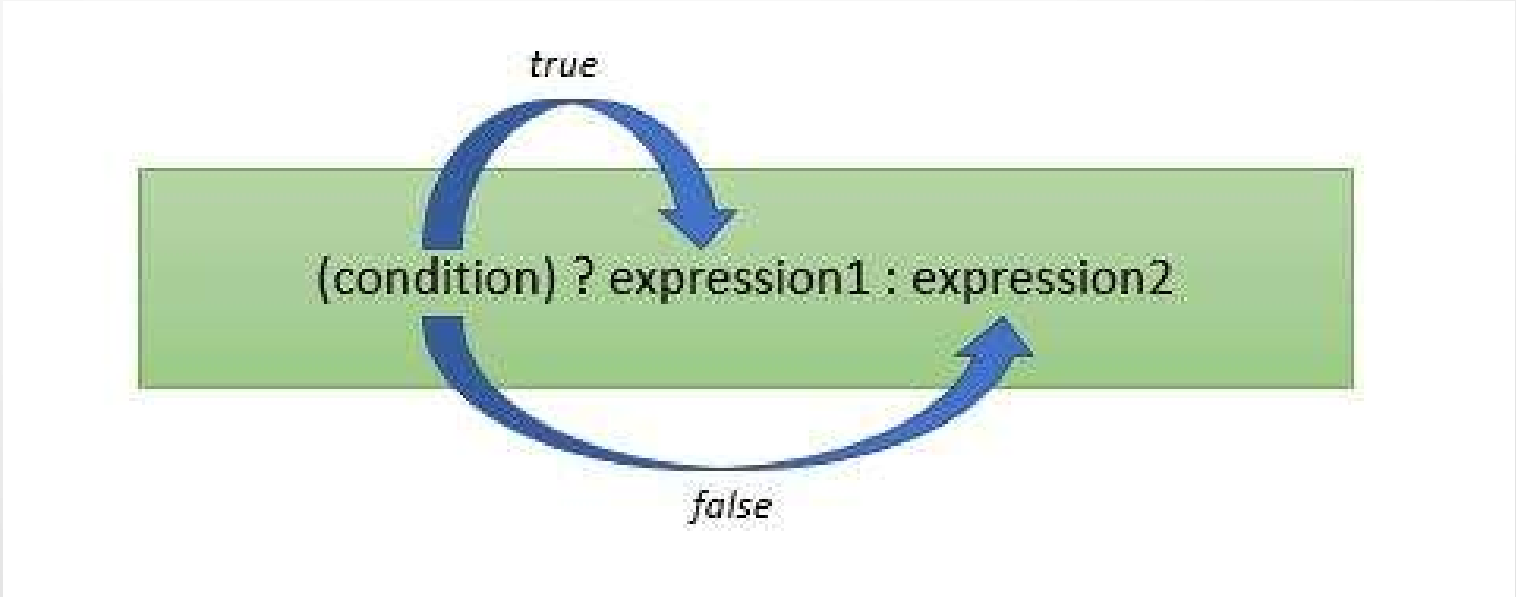
Conditional Operator

- The conditional operator (? :) is a ternary operator (it takes three operands). The conditional operator works as follows:
- The first operand is implicitly converted to bool. It is evaluated and all side effects are completed before continuing.
- If the first operand evaluates to true (1), the second operand is evaluated.
- If the first operand evaluates to false (0), the third operand is evaluated.



Conditional Operator

```
#include <iostream>
using namespace std;
int main() {
   int i = 1, j = 2;
   cout << ( i > j ? i : j ) << " is greater." << endl;
```



sizeof() Operator

- The sizeof is a keyword, but it is a compile-time operator that determines the size, in bytes, of a variable or data type.
- The sizeof operator can be used to get the size of classes, structures, unions and any other user defined data type.
- The syntax of using sizeof is as follows −
    - **sizeof (data type)**
- Where data type is the desired data type including classes, structures, unions and any other user defined data type.

sizeof() Operator

```
#include <iostream>
using namespace std;
int main() {
   cout << "Size of char : " << sizeof(char) << endl;
   cout << "Size of int : " << sizeof(int) << endl;
   cout << "Size of short int : " << sizeof(short int) << endl;
   cout << "Size of long int : " << sizeof(long int) << endl;
   cout << "Size of float : " << sizeof(float) << endl;
   cout << "Size of double : " << sizeof(double) << endl;
   return 0;
}
```

sizeof() Operator

O/P :

Size of char : 1

Size of int : 4

Size of short int : 2

Size of long int : 4

Size of float : 4

Size of double : 8

Comma Operator

- The purpose of comma operator is to string together several expressions.
- The value of a comma-separated list of expressions is the value of the right-most expression.
- Essentially, the comma's effect is to cause a sequence of operations to be performed.
- The values of the other expressions will be discarded.
- This means that the expression on the right side will become the value of the entire comma-separated expression.
- For example −
- **var = (count = 19, incr = 10, count+1);**

Comma Operator

```
#include <iostream>

using namespace std;

int main() {

   int i, j;

   j = 10;

   i = (j++, j+100, 999+j);

   cout << i;

   return 0;

}
```

When the above code is compiled and executed, it produces the following result − **1010**

Here is the procedure how the value of i gets calculated: j starts with the value 10. j is then incremented to 11. Next, j is added to 100. Finally, j (still containing 11) is added to 999, which yields the result 1010.

Operator precedence levels and Associativity

- If there are multiple operators in a single expression, the operations are not evaluated simultaneously. Rather, operators with higher precedence have their operations evaluated first.
- Let us consider an example:

  int x = 5 - 17 * 6;
- Here, the multiplication operator * is of higher level precedence than the subtraction operator -. Hence, 17 * 6 is evaluated first.
- As a result, the above expression is equivalent to

  int x = 5 - (17 * 6);

- If we wish to evaluate 5 - 17 first, then we must enclose them within parentheses: int x = (5 - 17) * 6;

Operator precedence levels and Associativity

```
#include <iostream>
using namespace std;
int main() {
  // evaluates 17 * 6 first
  int num1 = 5 - 17 * 6;
  // equivalent expression to num1
  int num2 = 5 - (17 * 6);
  // forcing compiler to evaluate 5 - 17 first
  int num3 = (5 - 17) * 6;
  cout << "num1 = " << num1 << endl;
  cout << "num2 = " << num2 << endl;
  cout << "num3 = " << num3 << endl;
  return 0;
}
```

Operator precedence levels and Associativity

Output :

num1 = -97

num2 = -97

num3 = -72