Department of Networks

First Year

Problem Solving and Programming 1

Operators Precedence in C++

- Operator precedence determines the grouping of terms in an expression.
- This affects evaluation of an expression and affects the result.
- 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 the first evaluation takes place for 3*2 and then 7 is added into it.
- 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 are evaluated first.

Operator	Associativity
() ++ - -	Left to right
* / %	Left to right
+ -	Left to right
<< >>	Left to right
< <= > >=	Left to right
== !=	Left to right
&	Left to right
^	Left to right
|	Left to right
&&	Left to right
||	Left to right
= += -= *= /= %=>>= <<= &= ^= |=	Right to Left

Example

- The following is an example of demonstrating operators precedence:

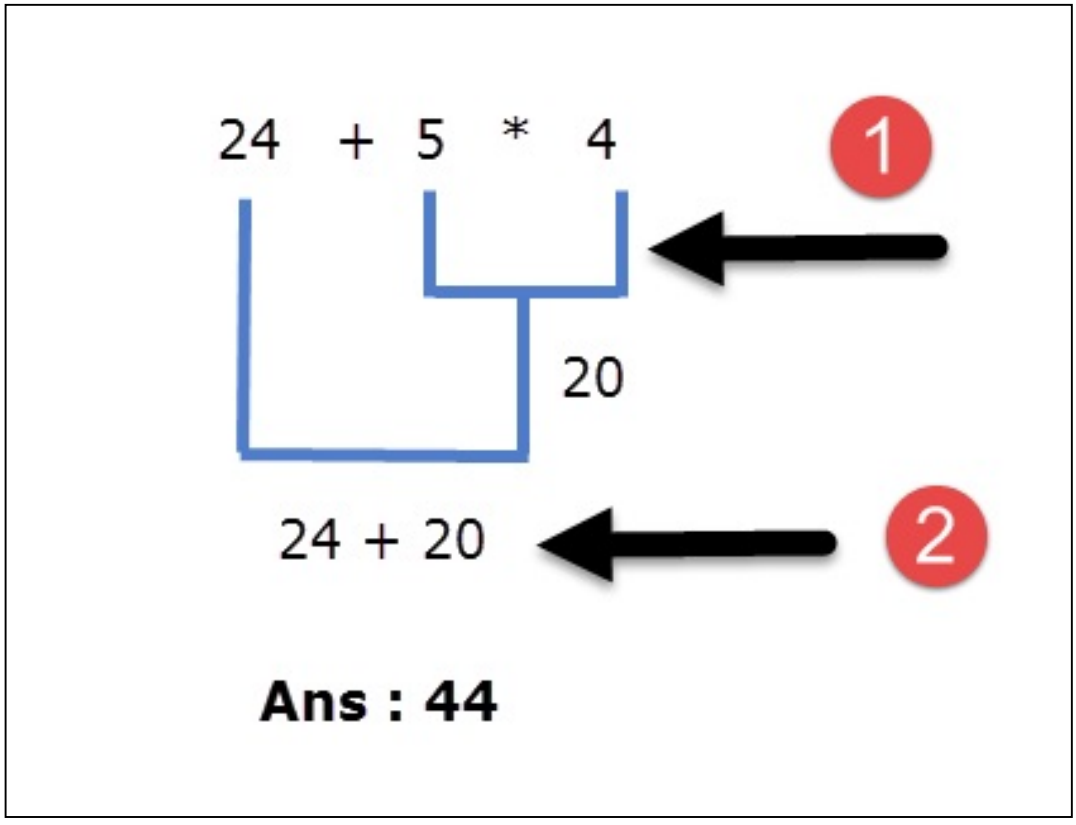
```
#include <iostream>
```

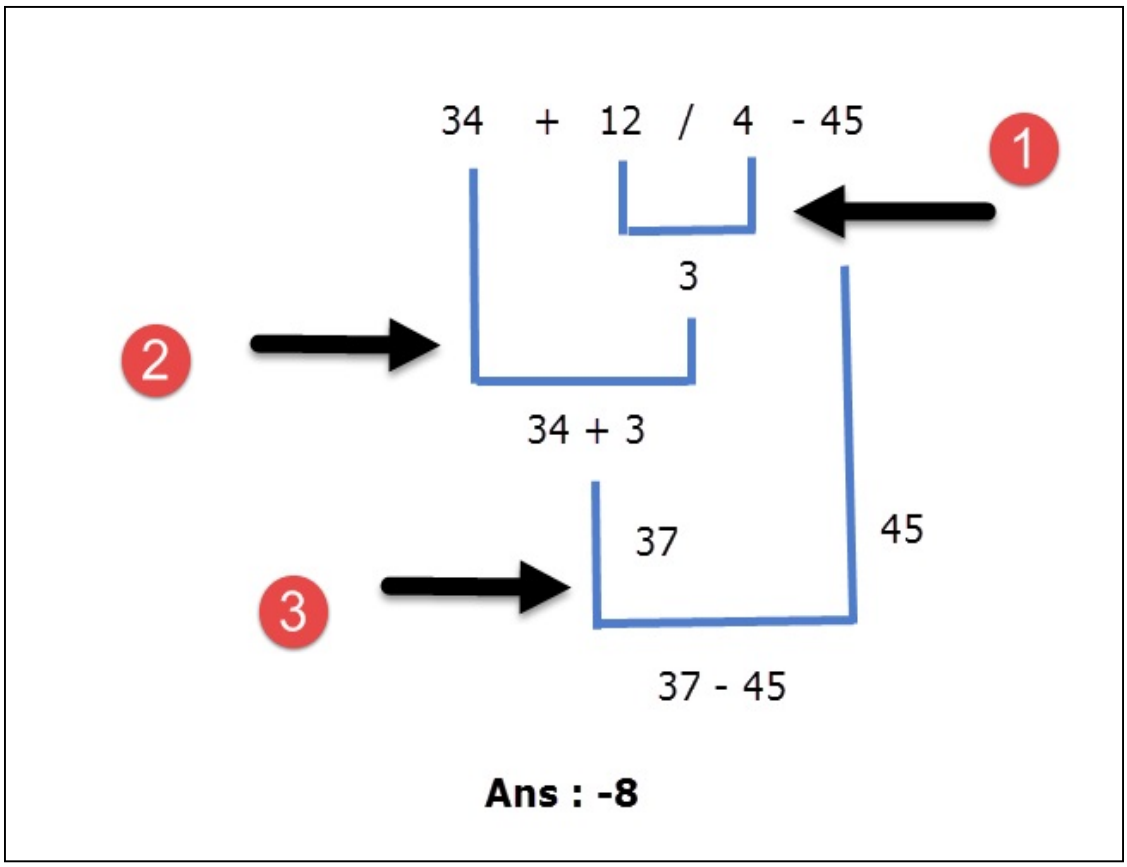
```
using namespace std;


main() {
 int a = 20;
   int b = 10;
   int c = 15;
   int d = 5;
   int e;


   e = a + b * c / d;       //    20 + 30
   cout << "Value of (a + b) * c / d is :" << e
<< endl ;


   e = ((a + b) * c) / d;    // (30 * 15 ) / 5
   cout << "Value of ((a + b) * c) / d is  :" <<
e << endl ;


   e = (a + b) * (c / d);   // (30) * (15/5)
   cout << "Value of (a + b) * (c / d) is  :" <<
e << endl ;
```

```
    e = a + (b * c) / d;      //  20 + (150/5)
    cout << "Value of a + (b * c) / d is  :" << e
<< endl ;


    return 0;
}
```

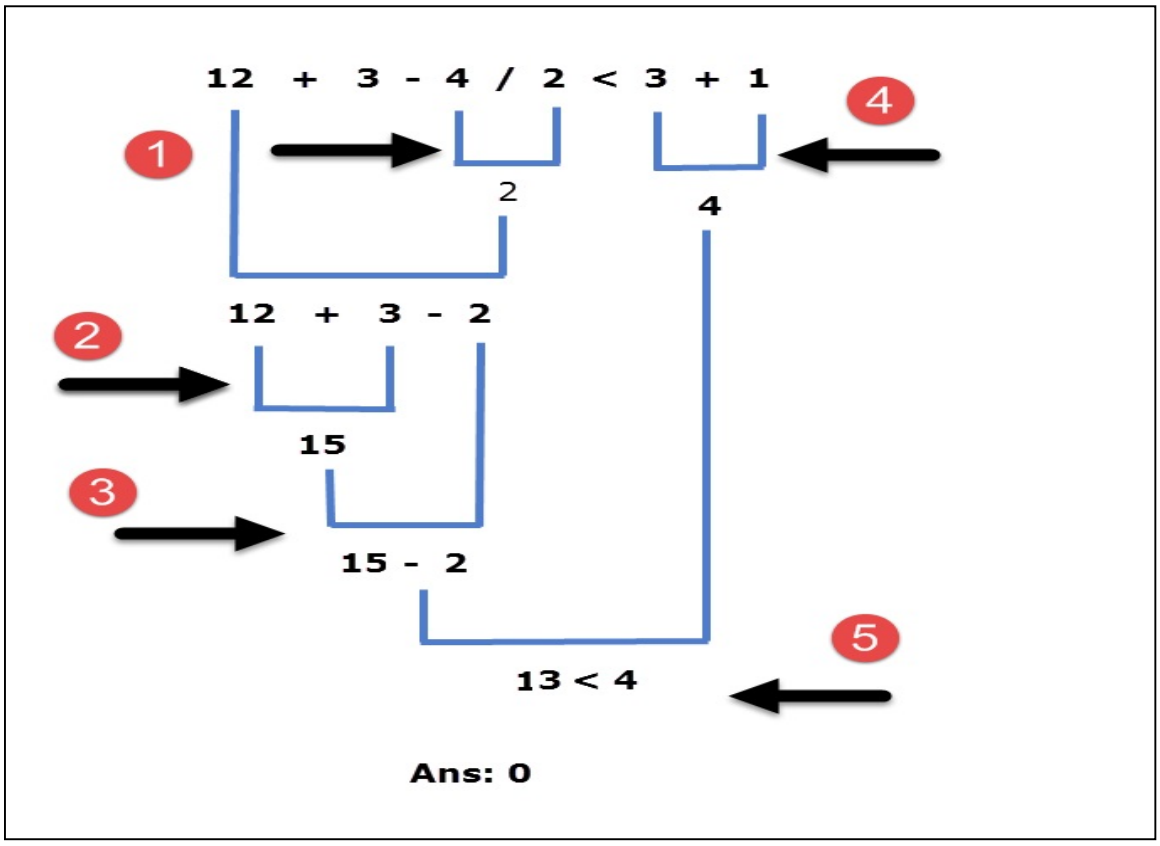
Output

```
Value of a + b * c / d is :50
Value of ((a + b) * c) / d is  :90
Value of (a + b) * (c / d) is  :90
Value of a + (b * c) / d is  :50
```





Ans : -8



Ans: 0

C++ Type Conversion

- Type conversion is converting one type of data to another type.
- It is also known as **Type Casting**. In C++, type casting has two forms:
  - Implicit type conversion: These conversions are performed by C++ in a type-safe manner. For example, conversions from smaller to larger integral types.
  - Explicit type conversion: These conversions are done explicitly by users using the pre-defined functions. Explicit conversions require a cast operator.

- The following example shows an explicit type conversion

```
double d = 5673.74;

int i;

 // cast double to int.

    i = (int)d;
```

```
cout<< i;
```

- The following example shows an implicit type conversion:

```
int num_int;

double num_double = 9.99;

// implicit conversion

// assigning a double value to an int variable

num_int = num_double;

cout << "num_int = " << num_int << endl;

cout << "num_double = " << num_double << endl;
```

- When the above code is compiled and executed, it produces the following result:

```
5673
```

- The following example shows an **implicit** type conversion

```
int int_num = 9;
long   long_num= int_num;
```

```
cout<< int_num <<endl;       // Outputs 9
cout<< long_num <<endl;    // Outputs 9
```

- The following example shows an **implicit** type conversion

```
// initializing a double variable
    double num_double = 3.56;
    cout << "num_double = " << num_double
<< endl;
   // C-style conversion from double to int
    int num_int1 = (int)num_double;
```

```
    cout << "num_int1    = " << num_int1 <<
endl;

//function-style conversion from double to
int

    int num_int2 = int(num_double);

    cout << "num_int2    = " << num_int2 <<
endl;
```