C++ operators: Arithmetic Assignment, Comparison, Logical

1-Operators

Once introduced to variables and constants, we can begin to operate with them by using *operators*. What follows is a complete list of operators. At this point, it is likely not necessary to know all of them, but they are all listed here to also serve as reference.

- **Assignment operator (=)**

The assignment operator assigns a value to a variable

X=5;

For example, let's have a look at the following code - I have included the evolution of the content stored in the variables as comments:

```
// assignment operator
#include <iostream>
using namespace std;
int main ()
{
  int a, b;         // a:?,  b:?
  a = 10;           // a:10, b:?
  b = 4;            // a:10, b:4
  a = b;            // a:4,  b:4
  b = 7;            // a:4,  b:7
  cout << "a:";
  cout << a;
  cout << " b:";
  cout << b;
}
```

Output:

a:4 b:7

- **C++ Arithmetic Operators**

Arithmetic Operators in C++ are used to perform arithmetic or mathematical operations on the operands. For example, '**+**' is used for addition, '**–**' is used for subtraction, '*****' is used for multiplication, etc. In simple terms, arithmetic operators are used to perform arithmetic operations on variables and data; they follow the same relationship between an operator and an operand.

C++ Arithmetic operators are of 2 types:

- **Unary Arithmetic Operator**
- **Binary Arithmetic Operator**

1. Binary Arithmetic Operator

These operators operate or work with two operands. C++ provides **5** *Binary Arithmetic Operators* for performing arithmetic functions:

Operator	Name of the Operators	Operation	Implementation
+	Addition	Used in calculating the Addition of two operands	x+y
–	Subtraction	Used in calculating Subtraction of two operands	x-y
*	Multiplication	Used in calculating Multiplication of two operands	x*y
/	Division	Used in calculating Division of two operands	x/y
%	Modulus	Used in calculating Remainder after	x%y

Operator	Name of the Operators	Operation	Implementation
		calculation of two operands	

```
// arithmetic function
#include <iostream>
using namespace std;
int main()
{
        int GFG1, GFG2;
        GFG1 = 10;
        GFG2 = 3;
        // printing the sum of GFG1 and GFG2
        cout<< "GFG1 + GFG2= " << (GFG1 + GFG2) << endl;
        // printing the difference of GFG1 and GFG2
        cout << "GFG1 - GFG2 = " << (GFG1 - GFG2) << endl;
        // printing the product of GFG1 and GFG2
        cout << "GFG1 * GFG2 = " << (GFG1 * GFG2) << endl;
        // printing the division of GFG1 by GFG2
        cout << "GFG1 / GFG2 = " << (GFG1 / GFG2) << endl;
        // printing the modulo of GFG1 by GFG2
        cout << "GFG1 % GFG2 = " << (GFG1 % GFG2) << endl;
        return 0;
}
```

Output

```
GFG1 + GFG2= 13
GFG1 - GFG2 = 7
GFG1 * GFG2 = 30
GFG1 / GFG2 = 3
GFG1 % GFG2 = 1
```

2. Unary Operator

These operators operate or work with a single operand.

Operator	Symbol	Operation	Implementation
Decrement Operator	—	Decreases the integer value of the variable by one	–x or x —
Increment Operator	++	Increases the integer value of the variable by one	++x or x++

Example:

C++

```
// C++ Program to demonstrate the
// increment and decrement operators
#include <iostream>
using namespace std;
int main()
{
   int x = 5;
   // This statement Incremented 1
   cout << "x++ is " << x++ << endl;
   // This statement Incremented 1
   // from already Incremented
   // statement resulting in
   // Incrementing of 2
   cout << "++x is " << ++x << endl;
   int y = 10;
   // This statement Decremented 1
   cout << "y-- is " << y-- << endl;
```

```
    // This statement Decremented 1
    // from already Decremented
    // statement resulting in
    // Decrementing of 2
    cout << "--y is " << --y << endl;

    return 0;
}
```

Output:

x++ is 5

++x is 7

y-- is 10

--y is 8

In **++x**, the variable's value is first increased/incremented before being utilised in the program.

In **x++**, a variable's value is assigned before it is increased/incremented.

Similarly happens for the decrement operator.

- **Comparison Operators**

Comparison operators are used to compare two values (or variables). This is important in programming, because it helps us to find answers and make decisions.

The return value of a comparison is either 1 or 0, which means **true** (1) or **false** (0). These values are known as **Boolean values**, and you will learn more about them in the Booleans and If..Else chapter.

In the following example, we use the **greater than** operator (>) to find out if 5 is greater than 3:

A list of all comparison operators:

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

- **Logical Operators**

As with comparison_operators, you can also test for **true** (1) or **false** (0) values with **logical operators**.

Logical operators are used to determine the logic between variables or values:

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

2- Operators' precedence

Cout<<10 + 5 *5 ;

```
int main()
{
cout << 10 + 5 * 5 << "\n";
// 5 * 5 = 25
// 10 + 25 = 35
```

```
ndowsDebugLauncher.exe --stdin=Microso
ft-MIEngine-In-twmziquv.bpk --stdout=M
icrosoft-MIEngine-Out-azgidj2s.1fz --s
tderr=Microsoft-MIEngine-Error-1cfzjhm
o.htp --pid=Microsoft-MIEngine-Pid-yu3
3hkxp.m23 --dbgExe=C:\\msys64\\mingw64
\\bin\\gdb.exe --interpreter=mi
35
```

Cout<<10 – 5 *5 ;

```
int main()
{
    cout << 10 + 5 * 5 << "\n";
    // 5 * 5 = 25
    // 10 + 25 = 35
    cout << 10 - 5 * 5 << "\n";
    // 5 * 5 = 25
    // 10 - 25 = -15
    return 0;
}
```

```
35

osama@Osama_Elzero MINGW64 ~/Desktop/Fundamentals
$  /usr/bin/env c:\\Users\\osama\\.vscode\\extensions\\ms-vscode.cpptools-1.13.9-win32-x64\\debugAdapters\\bin\\WindowsDebugLauncher.exe --stdin=Microsoft-MIEngine-In-bqvlhp0i.bo0 --stdout=Microsoft-MIEngine-Out-ukn4gj3m.kmq --stderr=Microsoft-MIEngine-Error-mqxna0g4.xtz --pid=Microsoft-MIEngine-Pid-oswtoun.tuw --dbgExe=C:\\msys64\\mingw64\\bin\\gdb.exe --interpreter=mi
35
-15
```

Cout <<20 / *4;

```
int main()
{
    cout << 10 + 5 * 5 << "\n";
    // 5 * 5 = 25
    // 10 + 25 = 35
    cout << 10 - 5 * 5 << "\n";
    // 5 * 5 = 25
    // 10 - 25 = -15
    cout << 20 / 5 * 4 << "\n";
    // 20 / 5 = 4
    // 4 * 4 = 16
    return 0;
}
```

```
-15

osama@Osama_Elzero MINGW64 ~/Desktop/Fundamentals
$  /usr/bin/env c:\\Users\\osama\\.vscode\\extensions\\ms-vscode.cpptools-1.13.9-win32-x64\\debugAdapters\\bin\\WindowsDebugLauncher.exe --stdin=Microsoft-MIEngine-In-5fd40mkq.xjf --stdout=Microsoft-MIEngine-Out-d2vu25ko.ryg --stderr=Microsoft-MIEngine-Error-htv0lhs0.vql --pid=Microsoft-MIEngine-Pid-jymcbypz.m1b --dbgExe=C:\\msys64\\mingw64\\bin\\gdb.exe --interpreter=mi
35
-15
16

osama@Osama_Elzero MINGW64 ~/Desktop/F
```

```
    cout << 10 + 20 / 5 * 4 << "\n";
    // 10 + 16 = 26
    // 20 / 5 = 4
    // 4 * 4 = 16
    return 0;
}
```

```
ft-MIEngine-In-lyhnje0t.wnn --stdout=Microsoft-MIEngine-Out-hmx5ionq.x2u --stderr=Microsoft-MIEngine-Error-xc13ir24.ci0 --pid=Microsoft-MIEngine-Pid-rdhivkl1.e2f --dbgExe=C:\\msys64\\mingw64\\bin\\gdb.exe --interpreter=mi
35
-15
16
26

osama@Osama_Elzero MINGW64 ~/Desktop/F
```