C# Bitwise and Bit Shift Operators

C# provides 4 bitwise and 2 bit shift operators.

Bitwise and bit shift operators are used to perform bit level operations on integer (int, long, etc) and boolean data. These operators are not commonly used in real life situations.

The bitwise and bit shift operators available in C# are listed below.

List of C# Bitwise Operators

Operator	Operator Name
~	Bitwise Complement
&	Bitwise AND
|	Bitwise OR
^	Bitwise Exclusive OR (XOR)
<<	Bitwise Left Shift
>>	Bitwise Right Shift

Bitwise OR

Bitwise OR operator is represented by `|`. It performs bitwise OR operation on the corresponding bits of two operands. If either of the bits is `1`, the result is `1`. Otherwise the result is `0`.

If the operands are of type `bool`, the bitwise OR operation is equivalent to logical OR operation between them.

For Example,

```
14 = 00001110 (In Binary)

11 = 00001011 (In Binary)
```

Bitwise `OR` operation between 14 and 11:

```
00001110

00001011

--------

00001111 = 15 (In Decimal)
```

Example 1: Bitwise OR

```
using System;

namespace Operator
{
        class BitWiseOR
        {
                public static void Main(string[] args)
                {
                        int firstNumber = 14, secondNumber = 11, result;
                        result = firstNumber | secondNumber;
                        Console.WriteLine("{0} | {1} = {2}", firstNumber,
secondNumber, result);
                }
        }
}
```

When we run the program, the output will be:

```
14 | 11 = 15
```

Bitwise AND

Bitwise AND operator is represented by `&`. It performs bitwise AND operation on the corresponding bits of two operands. If either of the bits is `0`, the result is `0`. Otherwise the result is `1`.

If the operands are of type `bool`, the bitwise AND operation is equivalent to logical AND operation between them.

For Example,

```
14 = 00001110 (In Binary)

11 = 00001011 (In Binary)
```

Bitwise AND operation between 14 and 11:

```
00001110

00001011

--------

00001010 = 10 (In Decimal)
```

Example 2: Bitwise AND

```
using System;

namespace Operator
{
        class BitWiseAND
        {
                public static void Main(string[] args)
                {
                        int firstNumber = 14, secondNumber = 11, result;
                        result = firstNumber & secondNumber;
                        Console.WriteLine("{0} & {1} = {2}", firstNumber,
secondNumber, result);
                }
        }
}
```

When we run the program, the output will be:

```
14 & 11 = 10
```

Bitwise XOR

Bitwise XOR operator is represented by `^`. It performs bitwise XOR operation on the corresponding bits of two operands. If the corresponding bits are **same**, the result is `0`. If the corresponding bits are **different**, the result is `1`.

If the operands are of type `bool`, the bitwise XOR operation is equivalent to logical XOR operation between them.

For Example,

```
14 = 00001110 (In Binary)

11 = 00001011 (In Binary)
```

Bitwise XOR operation between 14 and 11:

```
00001110

00001011

--------

00000101 = 5 (In Decimal)
```

Example 3: Bitwise XOR

```
using System;

namespace Operator
{
        class BitWiseXOR
```

```
        {
                public static void Main(string[] args)
                {
                        int firstNumber = 14, secondNumber = 11, result;
                        result = firstNumber^secondNumber;
                        Console.WriteLine("{0} ^ {1} = {2}", firstNumber,
secondNumber, result);
                }
        }
}
```

When we run the program, the output will be:

```
14 ^ 11 = 5
```

Bitwise Complement

Bitwise Complement operator is represented by `~`. It is a unary operator, i.e. operates on only one operand. The `~` operator **inverts** each bits i.e. changes 1 to 0 and 0 to 1.

For Example,

```
26 = 00011010 (In Binary)
```

Bitwise Complement operation on 26:

```
~ 00011010 = 11100101 = 229 (In Decimal)
```

Example 4: Bitwise Complement

```
using System;

namespace Operator
```

```
{
        class BitWiseComplement
        {
                public static void Main(string[] args)
                {
                        int number = 26, result;
                        result = ~number;
                        Console.WriteLine("~{0} = {1}", number, result);
                }
        }
}
```

When we run the program, the output will be:

```
~26 = -27
```

We got `-27` as output when we were expecting `229`. **Why did this happen?**
It happens because the binary value `11100101` which we expect to be `229` is actually a 2's complement representation of `-27`. Negative numbers in computer are represented in 2's complement representation.
For any integer n, 2's complement of `n` will be `-(n+1)`.

2's complement

Decimal	Binary	2's Complement
0	00000000	-(11111111 + 1) = -00000000 = -0 (In Decimal)
1	00000001	-(11111110 + 1) = -11111111 = -256 (In Decimal)
229	11100101	-(00011010 + 1) = -00011011 = -27

Overflow values are ignored in 2's complement.

The bitwise complement of `26` is 229 (in decimal) and the 2's complement of `229` is `-27`. Hence the output is `-27` instead of `229`.

Bitwise Left Shift

Bitwise left shift operator is represented by `<<`. The `<<` operator shifts a number to the left by a specified number of bits. Zeroes are added to the least significant bits.

In decimal, it is equivalent to

```
num * 2bits
```

For Example,

```
42 = 101010 (In Binary)
```

Bitwise Lift Shift operation on 42:

```
42 << 1 = 84 (In binary 1010100)

42 << 2 = 168 (In binary 10101000)

42 << 4 = 672 (In binary 1010100000)
```

Example 5: Bitwise Left Shift

```
using System;

namespace Operator
{
        class LeftShift
        {
                public static void Main(string[] args)
                {
                        int number = 42;

                        Console.WriteLine("{0}<<1 = {1}", number, number<<1);
                        Console.WriteLine("{0}<<2 = {1}", number, number<<2);
                        Console.WriteLine("{0}<<4 = {1}", number, number<<4);
                }
        }
```

```
}
```

When we run the program, the output will be:

```
42<<1 = 84
42<<2 = 168
42<<4 = 672
```

Bitwise Right Shift

Bitwise left shift operator is represented by `>>`. The `>>` operator shifts a number to the right by a specified number of bits. The first operand is shifted to right by the number of bits specified by second operand.

In decimal, it is equivalent to

```
floor(num / 2bits)
```

For Example,

```
42 = 101010 (In Binary)
```

Bitwise Lift Shift operation on 42:

```
42 >> 1 = 21 (In binary 010101)

42 >> 2 = 10 (In binary 001010)

42 >> 4 = 2 (In binary 000010)
```

Example 6: Bitwise Right Shift

```
using System;
```

```
namespace Operator
{
        class LeftShift
        {
                public static void Main(string[] args)
                {
                        int number = 42;

                        Console.WriteLine("{0}>>1 = {1}", number, number>>1);
                        Console.WriteLine("{0}>>2 = {1}", number, number>>2);
                        Console.WriteLine("{0}>>4 = {1}", number, number>>4);
                }
        }
}
```

When we run the program, the output will be:

```
42>>1 = 21
42>>2 = 10
42>>4 = 2
```