C# .Net Programming Assignment 8

•Create separate visual Studio project for each problem statement separately.
•For Business logic write separate class.
•Use Object Oriented concepts while writing the program.

1.There is one abstract class named as MarvellousNumber which contains one characteristics as integer And multiple abstract behaviours as

int CountOnBit()
Count number of 1's in given number

void DisplayBinary()
Display binary representations of given number

boolean CheckBit(int pos)
Check bit at specified position is on or off

void OffBit(int pos)
Off the bit at specified position if it is on

void ToggleBit(int pos)
Toggle the bit at specified position

We have to design one another class Bitwise named as which provide definitions for all abstract methods.

```
using System;

abstract class MarvellousNumber
{
        public int no;

        public MarvellousNumber(int value)
        {
                value = no;
        }

        public abstract int CountOnBit();
        public abstract void DisplayBinary();
        public abstract boolean CheckBit(int pos);
        public abstract void OffBit(int pos);
        public abstract void ToggleBit(int pos);
}
```

```
class Bitwise : MarvellousNumber
{
        public Bitwise(int value) : base(value)
        {
        }

        public override int CountOnBit()
        {
                // Logic
        }

        public override void DisplayBinary()
        {
                // Logic
        }

        public override boolean CheckBit(int pos)
        {
                // Logic
        }

        public override void OffBit(int pos)
        {
                // Logic
        }

        public override void ToggleBit(int pos)
        {
                // Logic
        }
}

public class Marvellous
{
        public static void Main()
        {
                Bitwise obj = new Bitwise(735);

                Console.WriteLine("Number of 1 {0}",obj.CountOnBit());

                Console.WriteLine("Binary representation is - ");
                obj.DisplayBinary();
```

```
		if(obj.CheckBit(5))
		{
			Console.WriteLine("Bit at 5th position is on");
		}
		else
		{
			Console.WriteLine("Bit at 5th position is off");
		}

		// Off the 7th bit of number
		obj.OffBit(7);
		Console.WriteLine("Updated number is {0}",obj.no);

		// Toggle the bit at 6th position
		obj.ToggleBit(6);
		Console.WriteLine("Updated number is {0}",obj.no);
	}
}
```

Above application is purely based on bitwise operations.

Please refer below details before solving the assignment.

A bitwise operator is an operator used to perform bitwise operations on bit patterns or binary numerals that involve the manipulation of individual bits.

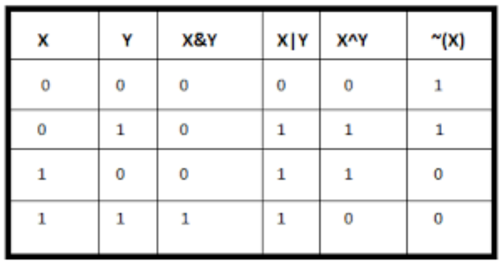
Bitwise operators are used in:

- Communication stacks where the individual bits in the header attached to the data signify important information
- Embedded software for controlling different functions in the chip and indicating the status of hardware by manipulating the individual bits of hardware registers of embedded micro controllers
- Low-level programming for applications such as device drivers, cryptographic software, video decoding software, memory allocators, compression software and graphics
- Maintaining large sets of integers efficiently in search and optimisation problems
- Bitwise operations performed on bit flags, which can enable an instance of enumeration type to store any combination of values defined in an enumerator list

Unlike common logical operators (like +, -, *), which work with bytes or groups of bytes, bitwise operators can check or set each of the individual bits within a byte.

Bitwise operators never cause overflow because the result produced after the bitwise operation is within the range of possible values for the numeric type involved.

The bitwise operators used in the C# are:



X	Y	X&Y	X|Y	X^Y	~(X)
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

- OR (|): Result is true if any of the operands is true.

- AND (&): Result is true only if both operands are true.

It can be used to set up a mask to check the values of certain bits.

- XOR (^): Result is true only if one of its operands is true.

It is used mainly to toggle certain bits. It also helps to swap two variables without using a third one.

- Bitwise Complement or Inversion or NOT (~): Provides the bitwise complement of an operand by inverting its value such that all zeros are turned into ones and all ones are turned to zeros.

- >> (Right-Shift) and << (Left-Shift) Operator: Moves the bits the number of positions specified by the second operand in the right or left direction.

While the right-shift operation is an arithmetic shift for operands of type int or long, it is a logical shift for operands of type uint or ulong. Shift operators are used in aligning bits.