Ex. No. 2	OPERATOR OVERLOADING		
Date of Exercise	20.07.2016	Date of Upload	23.08.2016

Aim

To write a Program in C# to overload various operators such as arithmetic, comparison and further user defined casting for the **Matrix Application.**

Description

Syntax of operator overloading

We can redefine or overload most of the built-in operators available in C#. Thus a programmer can use operators with user-defined types as well. Overloaded operators are functions with special names the keyword **operator** followed by the symbol for the operator being defined. similar to any other function, an overloaded operator has a return type and a parameter list.

For example:

```
public static Box operator+ (Box b, Box c)
{
   Box box = new Box();
   box.length = b.length + c.length;
   box.breadth = b.breadth + c.breadth;
   box.height = b.height + c.height;
   return box;
}
```

The above function implements the addition operator (+) for a user-defined class Box. It adds the attributes of two Box objects and returns the resultant Box object.

Rules to overload comparison operator

Overloadable and Non-Overloadable Operators

The following table describes the overload ability of the operators in C#:

Operators	Description
+, -, !, ~, ++, --	These unary operators take one operand and can be overloaded.
+, -, *, /, %	These binary operators take one operand and can be overloaded.
==, !=, <, >, <=, >=	The comparison operators can be overloaded
&&, ||	The conditional logical operators cannot be overloaded directly.
+=, -=, *=, /=, %=	The assignment operators cannot be overloaded.
=, ., ?:, ->, new, is, sizeof, typeof	These operators cannot be overloaded.

User - Defined Casts

C# allows two different types of casts : **Implicit** and **Explicit**

```
int I = 3;
long l = I; // implicit
short s = (short)I; // explicit
```

Explicit Casts are required where there is a risk that the cast might fail or some data might be lost. The following are some examples:

1. When converting from an int to a short , the short might not be large enough to hold the value of the int .

2. When converting from signed to unsigned data types, incorrect results will be returned if the signed variable holds a negative value.
3. When converting from floating - point to integer data types, the fractional part of the number will be lost.
4. When converting from a nullable type to a non - nullable type, a value of null will cause an exception.

C# support casts to and from own data types (struct and class)

- define a cast as a member operator of one of the relevant classes
- cast operator must be marked as either implicit or explicit to indicate how you are intending it to be used
- If you know that the cast is always safe whatever the value held by the source variable, then you define it as implicit .
- If, however, you know there is a risk of something going wrong for certain values — perhaps some loss of data or an exception being thrown - then you should define the cast as explicit

```
public static implicit operator float (Currency value)
{
// processing
}
```

- The cast defined here allows to implicitly convert the value of a Currency into a float
- If a conversion has been declared as implicit , the compiler will permit its use either implicitly or explicitly.
- If it has been declared as explicit , the compiler will only permit it to be used explicitly

Program

```
using System;
using System.Collections.Generic;
```

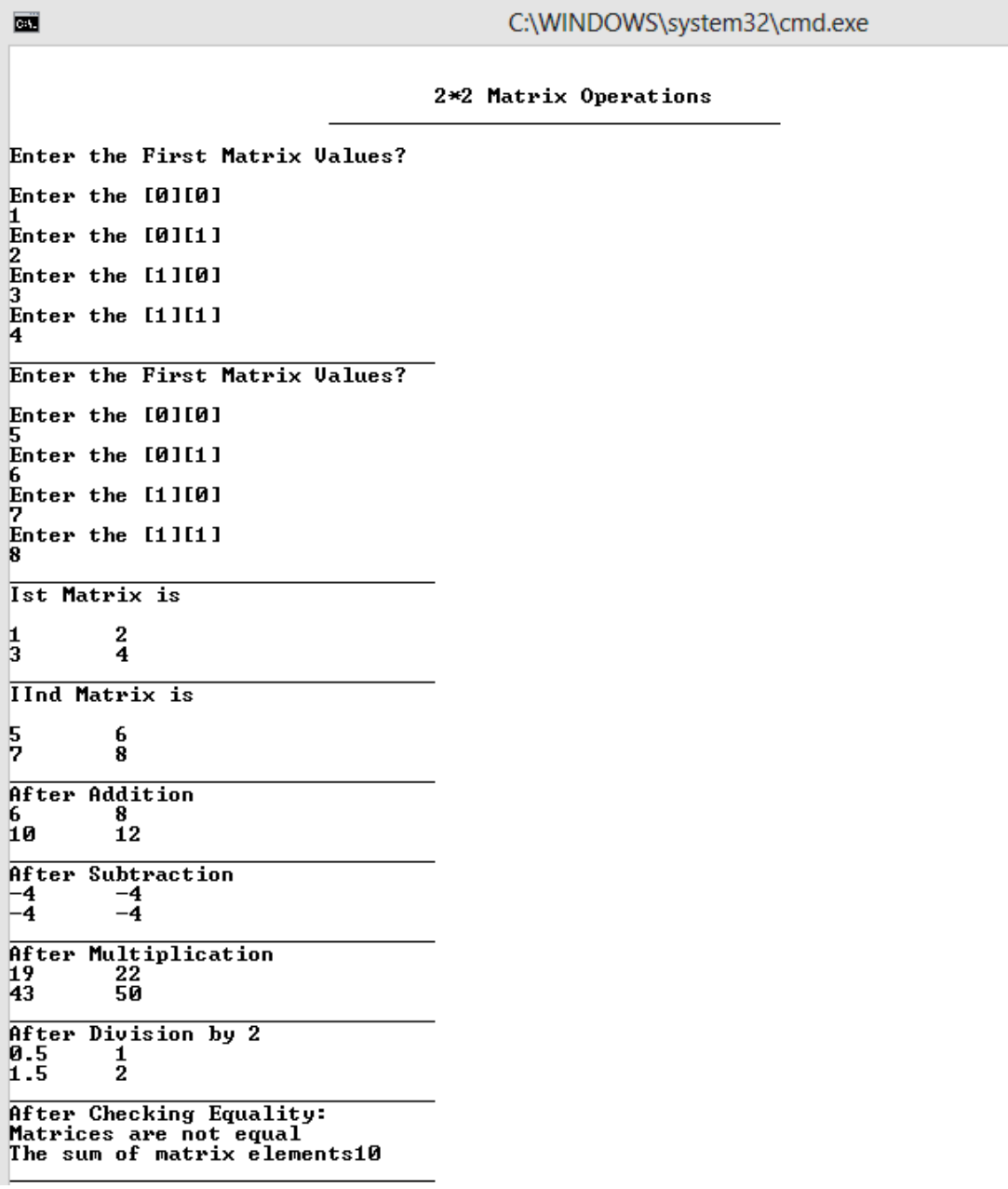
```
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MatrixOperationsOverloading
{
    class Matrix
    {
        public int[,] Mat1 = new int[2, 2];

        public int[,] Mat2 = new int[2, 2];

        public float[,] Mat3 = new float[2, 2];

        public int[,] Matadd = new int[2, 2];
        public int[,] Matsub = new int[2, 2];
        public int[,] Matmul = new int[2, 2];
        public float[,] Matdiv = new float[2, 2];

        public static int[,] operator +(Matrix obj, int[,] Mat2)
        {
            int[,] Matx=new int[2,2];
            for (int i = 0; i < 2; i++)
            {
                for (int j = 0; j < 2; j++)
                {
                    Matx[i, j] = obj.Mat1[i,j]+Mat2[i,j];
                }
            }
            return Matx;
        }

        public static int[,] operator -(Matrix obj, int[,] Mat2)
        {
            int[,] Matx = new int[2, 2];
            for (int i = 0; i < 2; i++)
            {
                for (int j = 0; j < 2; j++)
                {
                    Matx[i, j] = obj.Mat1[i, j] - Mat2[i, j];
                }
            }
            return Matx;
        }

        public static int[,] operator *(Matrix obj, int[,] Mat2)
        {
            int[,] Matx = new int[2, 2];
            int c, d, k,sum=0;

            for (c = 0; c < 2; c++)
            {
                for (d = 0; d < 2; d++)
                {
                    for (k = 0; k < 2; k++)
                    {
                        sum = sum + obj.Mat1[c,k] * obj.Mat2[k,d];
```

```
                }

                Matx[c,d] = sum;
                sum = 0;
            }
        }
        return Matx;
    }

    public static float[,] operator /(Matrix obj, int x)
    {
        float[,] Matx = new float[2, 2];
        for (int i = 0; i < 2; i++)
        {
            for (int j = 0; j < 2; j++)
            {
                obj.Mat3[i,j] = (float)obj.Mat1[i,j];
                Matx[i, j] = obj.Mat3[i, j] / x;
            }
        }
        return Matx;
    }

    public static bool operator ==(Matrix obj, int[,] Mat2)
    {
        int count = 0;
        for (int i = 0; i < 2; i++)
        {
            for (int j = 0; j < 2; j++)
            {
                if (obj.Mat1[i, j] == Mat2[i, j]) { count++; }
            }
        }
        if (count == 4) { return true; } else { return false; }
    }

    public static bool operator !=(Matrix obj, int[,] Mat2)
    {
        return !(obj==Mat2);
    }

    public static implicit operator float(Matrix obj)
    {
        float f = 0;
        for(int i = 0;i< 2;i++)
        {
            for(int j=0;j<2;j++)
            {
                f = f + obj.Mat1[i, j];
            }
        }
        return f;
    }

    static void Main(string[] args)
    {
        Matrix matobj = new Matrix();
```

```
            matobj.Initialisation();
            matobj.SetValues(matobj.Mat1);
            matobj.SetValues(matobj.Mat2);
            Console.WriteLine("Ist Matrix is\n");
            matobj.GetValues(matobj.Mat1);
            Console.WriteLine("IInd Matrix is\n");
            matobj.GetValues(matobj.Mat2);
            matobj.Matadd = matobj + matobj.Mat2;
            Console.WriteLine("After Addition");
            matobj.GetValues(matobj.Matadd);
            matobj.Matadd = matobj - matobj.Mat2;
            Console.WriteLine("After Subtraction");
            matobj.GetValues(matobj.Matadd);
            matobj.Matadd = matobj * matobj.Mat2;
            Console.WriteLine("After Multiplication");
            matobj.GetValues(matobj.Matadd);
            matobj.Matdiv = matobj / 2;
            Console.WriteLine("After Division by 2");
            matobj.GetValues(matobj.Matdiv);
            bool check = matobj == matobj.Mat2;
            Console.WriteLine("After Checking Equality:");
            if (check) { Console.WriteLine("Matrices are equal"); } else {
Console.WriteLine("Matrices are not equal"); }
            float f = matobj;
            Console.WriteLine("The sum of matrix elements" + f);
            Console.WriteLine("_____________________________________");
        }

        public void Initialisation()
        {
            Console.WriteLine("\n*___*___*___*___*___***__Welcome to the Matrix
Multiplication Program__***___*___*___*___*___*\n\n");
            Console.WriteLine("\t\t\t\t2*2 Matrix Operations");
            Console.WriteLine("\t\t\t_____________________________________\n");
        }

        public void SetValues(int[,] Mat3)
        {
            Console.WriteLine("Enter the First Matrix Values?\n");
            for (int i = 0; i < 2; i++)
            {
                for (int j = 0; j < 2; j++)
                {
                    Console.WriteLine("Enter the [{0}][{1}]",i,j);
                    Mat3[i,j] = Convert.ToInt32(Console.ReadLine());
                }
            }
            Console.WriteLine("_____________________________________");
        }

        public void GetValues(int[,] array) {
            for (int x = 0; x < array.GetLength(0); x += 1)
            {
                for (int y = 0; y < array.GetLength(1); y += 1)
                {
                    Console.Write(array[x, y]+"\t");
                }
```

```
                Console.WriteLine();
            }
            Console.WriteLine("_____________________________________");
        }

        public void GetValues(float[,] array)
        {
            for (int x = 0; x < array.GetLength(0); x += 1)
            {
                for (int y = 0; y < array.GetLength(1); y += 1)
                {
                    Console.Write(array[x, y] + "\t");
                }
                Console.WriteLine();
            }
            Console.WriteLine("_____________________________________");
        }

    }
}
```

Output



Result

The above program for operating overloading is compiled successfully and the screenshots are well described with successful outputs and constraints.

[Dr. S.P. Jeno Lovesum