NAME: KAYODE PETER TEMITOPE

MATRIC NUMBER: 208077

DEPARTMENT: COMPUTER SCIENCE (200 Level)

COURSE: OBJECT ORIENTED PROGRAMMING (CSC 235)

Assignment

1. Briefly explain with examples Strings, Inheritance, Polymorphism, operator overloading and interface operation in C#.
2. Present the operation of recursive in clear and concise manner using mathematical factorial example.

Solution to Question 1

A. Strings in C#: string is an object of Sysytem.String class that represent sequence of characters. It is also a keyword. Many operations can be performed on strings in C# such as:

a. Concatenation
b. Comparison
c. Getting
d. Substring
e. Search
f. Trim
g. Replacement etc.

Some string method names and their description:

a. Clone(): It is used to return a reference to this instance of String.
b. Compare(String, String): It is used to compares two specified String objects. It returns an integer that indicates their relative position in the sort order.
c. CompareOrdinal(String, String): It is used to compare two specified String objects by evaluating the numeric values of the corresponding Char objects in each string..
d. CompareTo(String): It is used to compare this instance with a specified String object. It indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified string.
e. Concat(String, String): It is used to concatenate two specified instances of String.
f. Contains(String): It is used to return a value indicating whether a specified substring occurs within this string.
g. Copy(String): It is used to create a new instance of String with the same value as a specified String.

Example:

```
using System;

namespace StringExample

{

   class Program

   {

       static void Main(string[] args)

       {

            String s1 = “Hello”;

            Console.WriteLine(s1);

       }

   }

}
```

Output

```
Hello
```

B. **Inheritance in C#:** inheritance is a process in which one object acquires all properties and behaviors of its parent object automatically. The class which inherit the members of another class is called “derived class” and the class whose members are inherited is called “base class”. The derived class is the specialized class for the base class.
Inheritance in C# can either be Single level Inheritance or Multi level Inheritance.

1. **Single Level Inheritance:** When one class inherits another class.
Example;

```
using System;

   public class Animal

    {

       public void eat() { Console.WriteLine("Sleeping"); }

   }

   public class Dog: Animal

   {
       public void bark() { Console.WriteLine("Playing"); }
   }
   class TestInheritance2{
       public static void Main(string[] args)
        {
           Dog d1 = new Dog();
           d1.eat();
           d1.bark();
        }
    }
```

Output:

```
Sleeping
Playing
```

2. **Multi Level Inheritace:** When one class inherits another class which is further inherited by another class. Inheritance is transitive so the last derived class acquires all the members of all its base classes.

   Example:

```
using System;
  public class Animal
   {
    public void eat() { Console.WriteLine("Eating"); }
  }
  public class Dog: Animal
  {
    public void bark() { Console.WriteLine("Barking"); }
  }
  public class BabyDog : Dog
  {
    public void weep() { Console.WriteLine("Weeping"); }
  }
  class TestInheritance2{
     public static void Main(string[] args)
      {
          BabyDog d1 = new BabyDog();
          d1.eat();
          d1.bark();
          d1.weep();
      }
   }
```

Output:

```
Eating
Barking
Weeping
```

C. **Polymorphism in C#:** Polymorphism is when an object has many forms or has one name with multiple functionalities. Polymorphism provides the ability to a class to have multiple implementations with the same name.
   Two types of polymorphism are:
   1. **Static or Compile-time polymorphism**: The decision of which method to be called is made at compile time. It is achieved by method overloading and operator overloading.
      Example:

```
using System;
namespace Polymorphism
{
   public class Calculate
   {
     public void AddNumbers(int a, int b)
     {
       Console.WriteLine("a + b = {0}", a + b);
     }
```

```
        public void AddNumbers(int a, int b, int c)
        {
            Console.WriteLine("a + b + c = {0}", a + b + c);
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Calculate c = new Calculate();
            c.AddNumbers(1, 2);
            c.AddNumbers(1, 2, 3);
            Console.WriteLine("\nPress Enter Key to Exit..");
            Console.ReadLine();
        }
    }
}
```

Output:

```
a + b + c = 6
Press Enter Key to Exit..
```

2. **Dynamic or Runtime polymorphism:** Here, the method name and method signature (number of parameters and parameter type must be the same and may have a different implementation.
   Example:

```
using System;
public class Animal
{
    public virtual void eat()
{
        Console.WriteLine("eating");
    }
}
public class Dog: Animal
{
    public override void eat()
    {
        Console.WriteLine("Eating bread");
    }
}
public class TestPolymorphism
{
    public static void Main()
    {
        Animal a= new Dog();
        a.eat();
    }
}
```

Output

```
Eating bread
```

D. **Operator Overloading in C#:** The mechanism of giving a special meaning to a standard C# operator with respect to a user defined data type such as classes or structures is known as operator overloading. A special function called operator function is used for overloading purpose. These special function or method must be public and static. They can take only value arguments. The ref and out parameters are not allowed as arguments to operator functions.
Example:

```
using System;
namespace OperatorOvlApplication
```

```
 {
   class Box
 {
     private double length;   // Length of a box
     private double breadth;  // Breadth of a box
     private double height;   // Height of a box

     public double getVolume() {
       return length * breadth * height;
     }
     public void setLength( double len ) {
       length = len;
     }
     public void setBreadth( double bre ) {
       breadth = bre;
     }
     public void setHeight( double hei ) {
       height = hei;
     }
     // Overload + operator to add two Box objects.
     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;
     }
   }
```

```
class Tester {
  static void Main(string[] args) {
    Box Box1 = new Box();   // Declare Box1 of type Box
    Box Box2 = new Box();   // Declare Box2 of type Box
    Box Box3 = new Box();   // Declare Box3 of type Box
    double volume = 0.0;    // Store the volume of a box here

    // box 1 specification
    Box1.setLength(6.0);
    Box1.setBreadth(7.0);
    Box1.setHeight(5.0);

    // box 2 specification
    Box2.setLength(12.0);
    Box2.setBreadth(13.0);
    Box2.setHeight(10.0);

    // volume of box 1
    volume = Box1.getVolume();
    Console.WriteLine("Volume of Box1 : {0}", volume);

    // volume of box 2
    volume = Box2.getVolume();
    Console.WriteLine("Volume of Box2 : {0}", volume);

    // Add two object as follows:
    Box3 = Box1 + Box2;

    // volume of box 3
```

```
        volume = Box3.getVolume();

        Console.WriteLine("Volume of Box3 : {0}", volume);

        Console.ReadKey();

      }

   }

}
```

Output

```
Volume of Box1 : 210

Volume of Box2 : 1560

Volume of Box3 : 5400
```

E. **Interface Operation in C#:** Interfaces are used along with classes to define what is known as a contract. A contract is an agreement on what the class will provide to an application. An interface declares the properties and method. It is up to the class to define exactly what the method will do.
   Example:

```
using System;

// A simple interface

interface inter1

{

    // method having only declaration

    // not definition

    void display();

}

// A class that implements interface.

class testClass : inter1

{

    // providing the body part of function

    public void display()
    {
        Console.WriteLine("Be a good boy");
    }
    // Main Method
    public static void Main (String []args)
    {
        // Creating object
        testClass t = new testClass();
        // calling method
        t.display();
    }
}
```

Output:

```
Be a good boy
```

Solution to Question 2

It is answered in the attached .cs file