Exercise 9: Multithreading and Synchronization

Multithreading

Multithreading allows a process to manage two or more concurrent threads. Each thread can handle a task independently. For example, while one thread performs a complex calculation, another can update the user interface, preventing the application from freezing.
C# Thread class provides properties and methods to create and control threads. It is found in System.Threading namespace.

Creating Threads without using ThreadStart delegate

For example, if we want execute a function called task1() through Thread, we can create a Thread without ThreadStart as given below

```
Thread t1 = new Thread(task1);
```

So, the above thread instance creation statement is implicitly converted to the ThreadStart delegate instance.
Now a new Thread is created called t1, it must be activated through an explicit method call via Start() in order bring the thread under control of CPU thread scheduling.

```
            t1.Start();
```

Additionally, Join() method can also be called to make the Main Thread to wait for the newly created threads.

```
using System;
using System.Threading;
class Program
 {
     public static void task1()
    {
        for(int i=1; i <= 10;i++)
        {
            Console.WriteLine(i);
            if(i == 5)
            {
                Thread.Sleep(5000);
            }
        }
    }
    public static void task2()
    {
        for (int i = 100; i <= 110; i++)
        {
            Console.WriteLine(i);
        }
    }
    static void Main(string[] args)
        {
          Thread t1 = new Thread(task1);
          Thread t2 = new Thread(task2);
```

```
            t1.Start();
            t2.Start();
            t1.Join();
            t2.Join();
            Console.ReadKey();
        }
 }
```

Synchronization in C#

One of the ways to achieve Synchronization in C# is by using the feature of lock, which locks the access to a block of code within the locked object. When a thread locks an object, no other thread can access the block of code within the locked object. Only when a thread releases the lock, then it is available for other threads to access it.

In C# Language, every object has a built-in lock. By using the feature of Synchronization, we can lock an object. Locking an object can be done by using the **lock** keyword, and the following is the syntax to use the lock.

```
lock(object)
{
    //Statement1
    //Statement2
    //And more statements to be synchronized
}
```

Complete All the Questions

1. Create a program to read a paragraph of text data and create 2 threads namely **word, vowel** and do the following operations with 2 seconds' delay.

A. word thread display each word one by one.

B. vowel thread prints each vowel one by one

2. Create a program to read 10 numbers and store in an array create 3 threads namely **even**, **odd** and do the following operations with 2 seconds' delay

A. even thread display all the even numbers in the array.

B. odd thread display all the odd numbers in the array.

Synchronization Question

3. Create a program to read three numbers as input and create 3 threads namely table1, table2 and table3 and display multiplication table without mixing any sequence with 2 seconds delay using Thread Synchronization.
