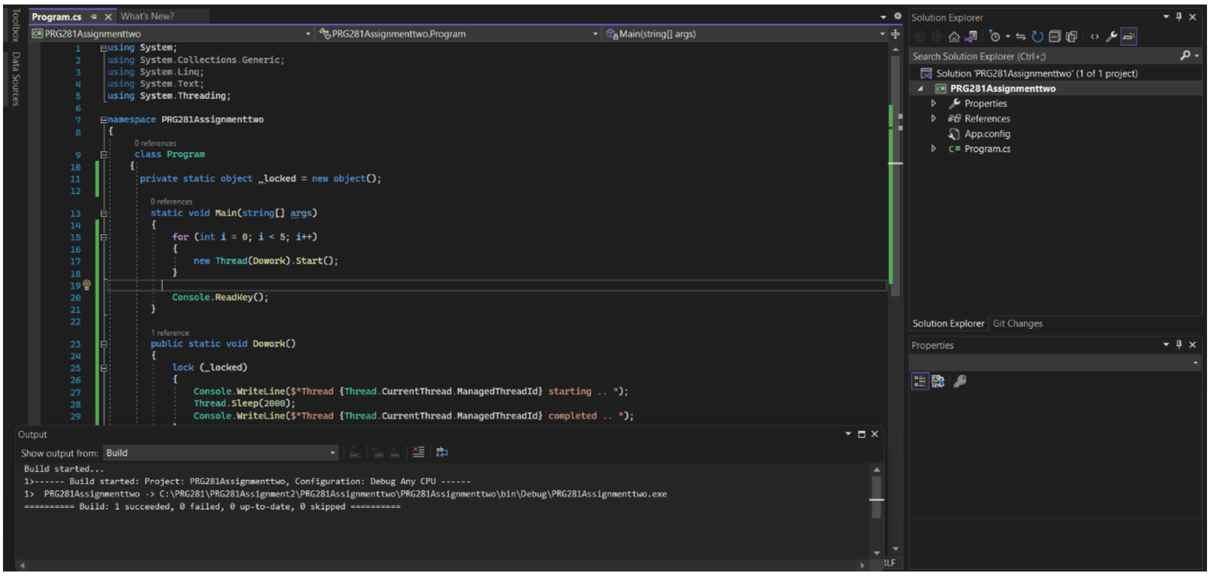
ASSIGNMENT 2

Student no : 577833

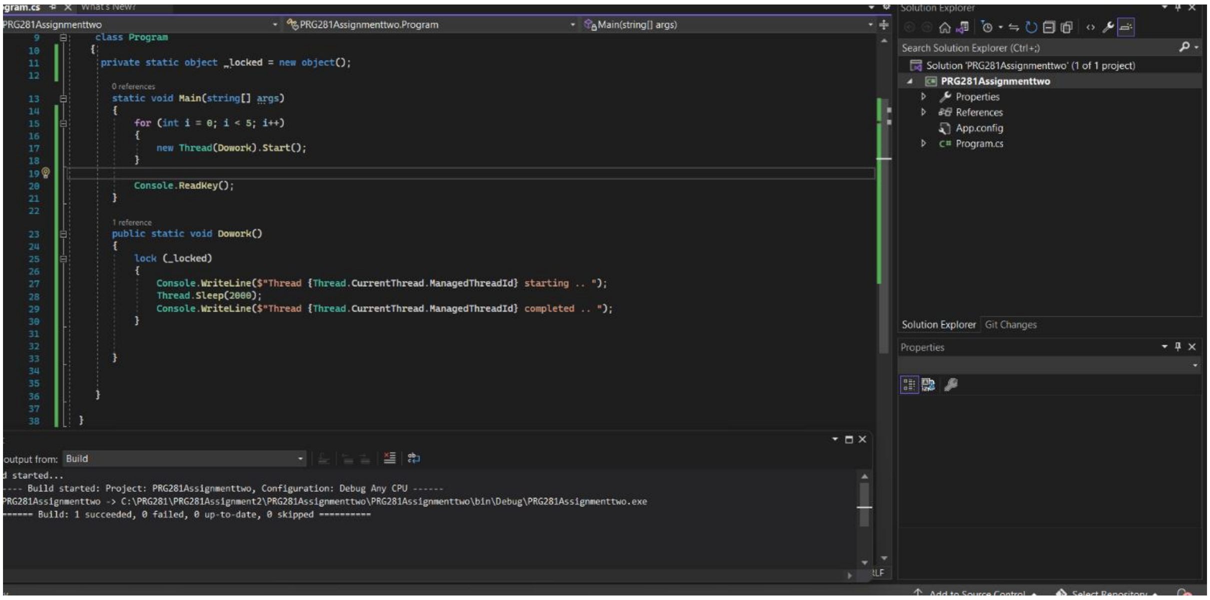
Full name : Hrudhay Reddy Garisa , lecturer name : Raymond Hood

Q1 (A) You would use the lock method in your program code to ensure that thread safety is being applied to your Object A , as object A has the critical action of the code . Lock method is a thread synchronization technique that allows each thread to execute the critical action of the code one at a time rather than executing the code concurrently which could cause errors . Like for example :   (CODELLIGENT, 2020)



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace PRG281Assignmenttwo
{
    0 references
    class Program
    {
     private static object _locked = new object();

        0 references
        static void Main(string[] args)
        {
            for (int i = 0; i < 5; i++)
            {
                new Thread(Dowork).Start();
            }

            Console.ReadKey();
        }

        1 reference
        public static void Dowork()
        {
            lock (_locked)
            {
                Console.WriteLine($"Thread {Thread.CurrentThread.ManagedThreadId} starting .. ");
                Thread.Sleep(2000);
                Console.WriteLine($"Thread {Thread.CurrentThread.ManagedThreadId} completed .. ");
```

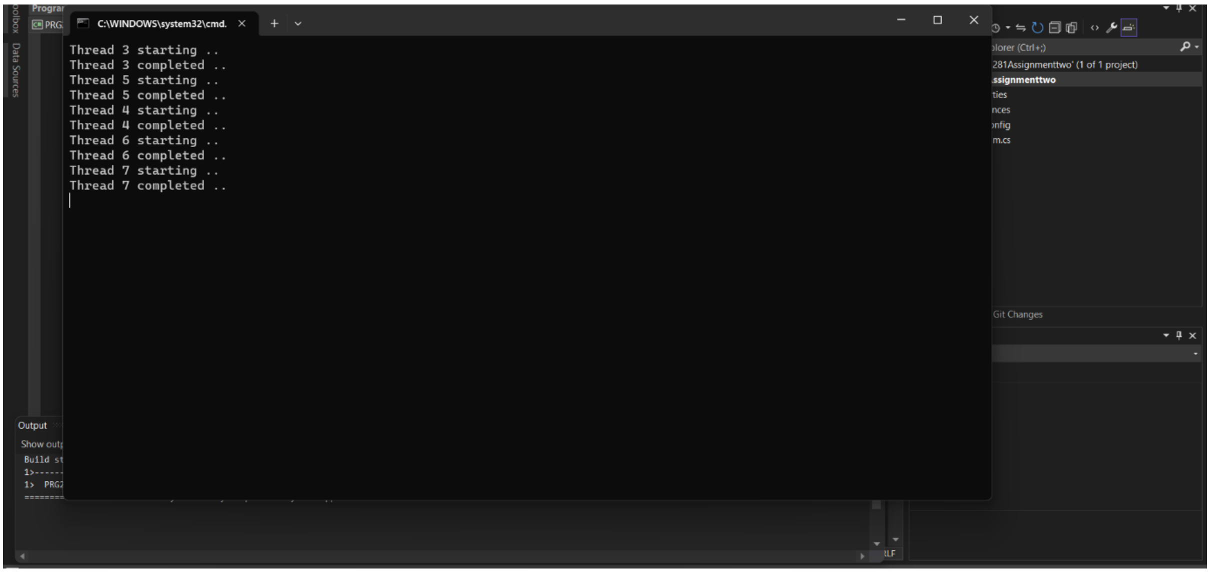
```
Build started...
1>------ Build started: Project: PRG281Assignmenttwo, Configuration: Debug Any CPU ------
1>  PRG281Assignmenttwo -> C:\PRG281\PRG281Assignment2\PRG281Assignmenttwo\PRG281Assignmenttwo\bin\Debug\PRG281Assignmenttwo.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```



```
    class Program
    {
     private static object _locked = new object();

        0 references
        static void Main(string[] args)
        {
            for (int i = 0; i < 5; i++)
            {
                new Thread(Dowork).Start();
            }

            Console.ReadKey();
        }

        1 reference
        public static void Dowork()
        {
            lock (_locked)
            {
                Console.WriteLine($"Thread {Thread.CurrentThread.ManagedThreadId} starting .. ");
                Thread.Sleep(2000);
                Console.WriteLine($"Thread {Thread.CurrentThread.ManagedThreadId} completed .. ");
            }



        }


    }

}
```

```
d started...
---- Build started: Project: PRG281Assignmenttwo, Configuration: Debug Any CPU ------
RG281Assignmenttwo -> C:\PRG281\PRG281Assignment2\PRG281Assignmenttwo\PRG281Assignmenttwo\bin\Debug\PRG281Assignmenttwo.exe
------ Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Output screen :



```
Thread 3 starting ..
Thread 3 completed ..
Thread 5 starting ..
Thread 5 completed ..
Thread 4 starting ..
Thread 4 completed ..
Thread 6 starting ..
Thread 6 completed ..
Thread 7 starting ..
Thread 7 completed ..
```

Q1 (C)

If you look at the code above , you will see that the lock method has enclosed the critical action of the code . The critical action of the code in this case is inside the Dowork() method , as the lock method will enable each thread to be synchronized meaning each thread will access the critical section of the code one at a time , meaning that the next thread can only start once the previous thread is finished , if you look at the example above in the output screen that thread 5 only starts once thread 3 is finished . (CODELLIGENT, 2020)

Q1 (B)

- Difficulty of debugging : It is much harder to fix an error in a multithreaded or application than it is to do so in a single-threaded . Therefore , it is more difficult to identify and verify root causes of errors when errors occur .
- Difficulty of coding : Multithreaded applications are not easy to write . Only experienced programmers should undertake coding for these types of applications .

- Difficulty of managing concurrency : The task of managing concurrency among threads is difficult and has the potential to introduce new problems into an application.

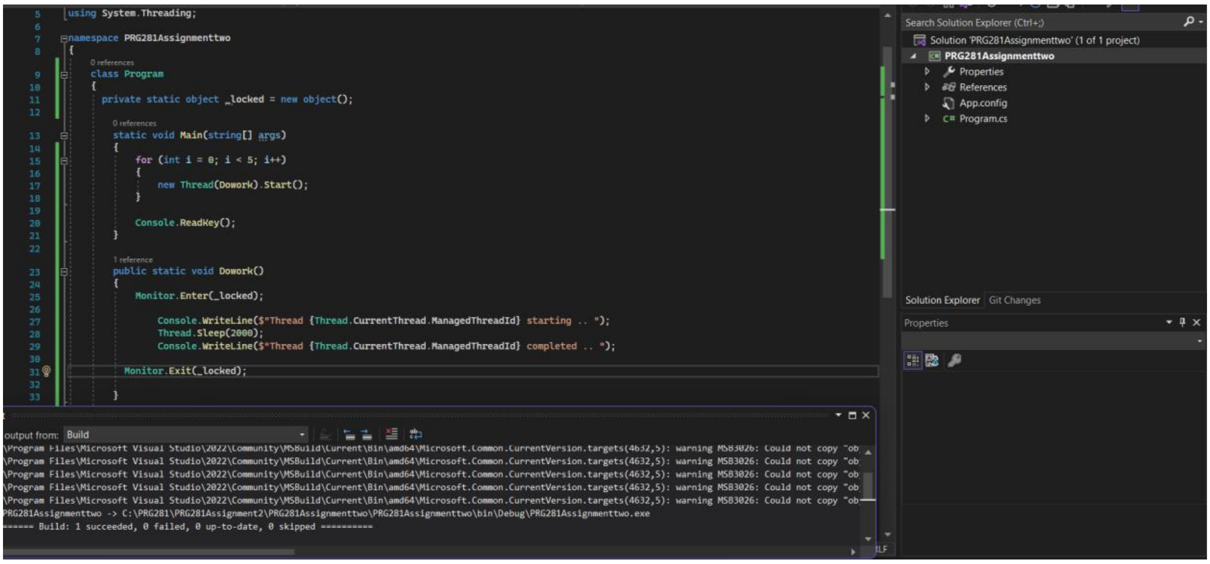
- Difficulty of testing : Testing a multithreaded application is more difficult than testing a single-threaded application because defects are often timing-related and more difficult to reproduce.

(Oracle, 2023)

Q1(D)

What is a thread safe data structure ? A thread safe data structure ensures that the shared data is protected from concurrent modifications, and that the operations are consistently performing in the way they were assigned to do . Thread safety can be achieved by using various techniques, such as Locking, Monitoring , Mutex , Semaphore and etc (Meikopoulous, 2021) .

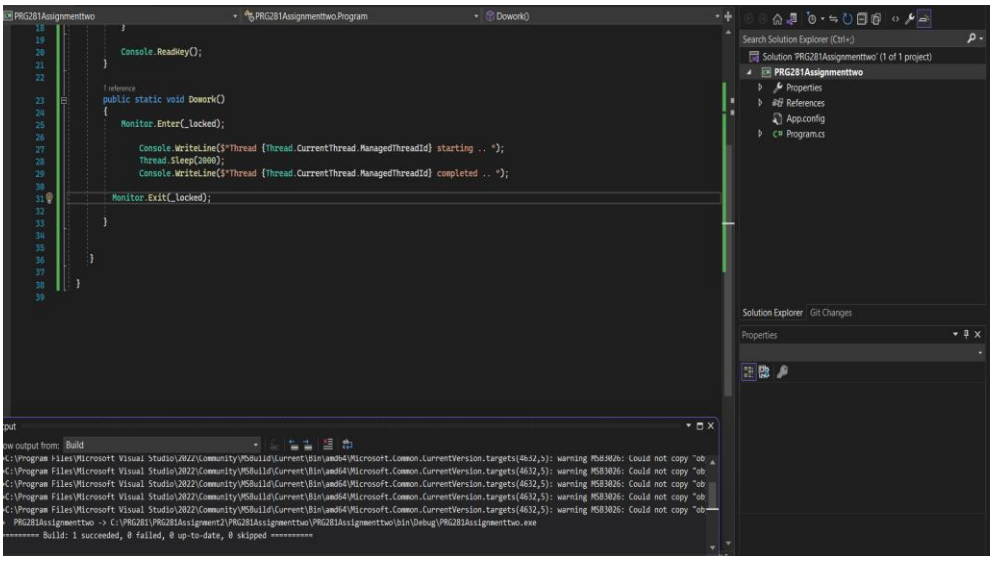
Examples of using Monitoring which is also a thread safe synchronization method :



```
using System.Threading;

namespace PRG281Assignmenttwo
{
    0 references
    class Program
    {
        private static object _locked = new object();

        0 references
        static void Main(string[] args)
        {
            for (int i = 0; i < 5; i++)
            {
                new Thread(Dowork).Start();
            }

            Console.ReadKey();
        }

        1 reference
        public static void Dowork()
        {
            Monitor.Enter(_locked);

                Console.WriteLine($"Thread {Thread.CurrentThread.ManagedThreadId} starting .. ");
                Thread.Sleep(2000);
                Console.WriteLine($"Thread {Thread.CurrentThread.ManagedThreadId} completed .. ");

          Monitor.Exit(_locked);

        }
```

output from: Build

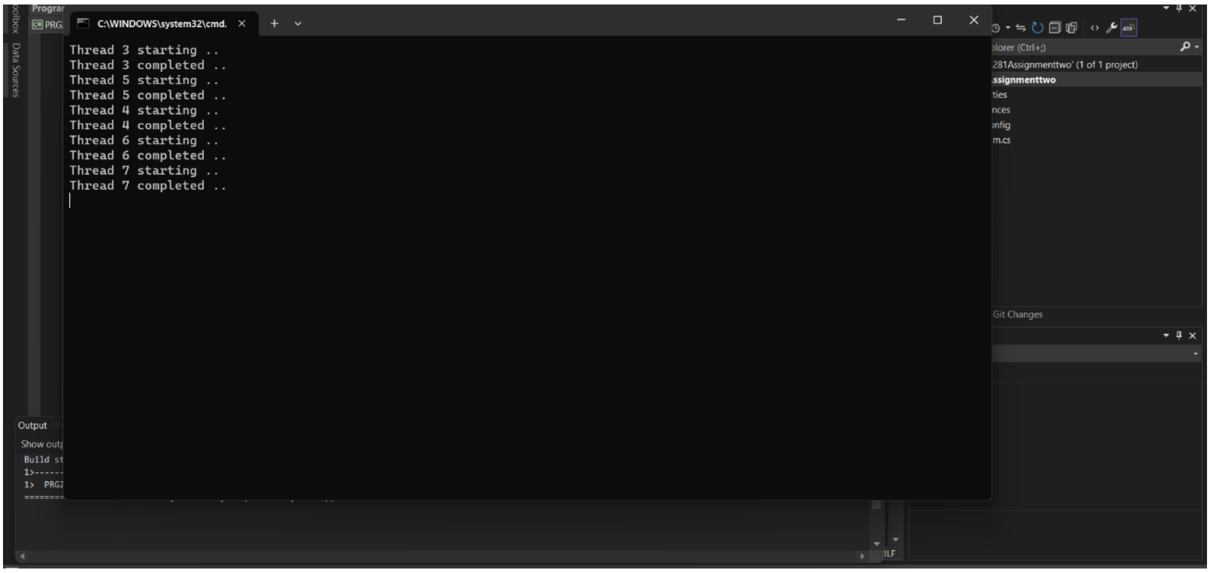
```
\Program Files\Microsoft Visual Studio\2022\Community\MSBuild\Current\Bin\amd64\Microsoft.Common.CurrentVersion.targets(4632,5): warning MSB3026: Could not copy "ob
\Program Files\Microsoft Visual Studio\2022\Community\MSBuild\Current\Bin\amd64\Microsoft.Common.CurrentVersion.targets(4632,5): warning MSB3026: Could not copy "ob
\Program Files\Microsoft Visual Studio\2022\Community\MSBuild\Current\Bin\amd64\Microsoft.Common.CurrentVersion.targets(4632,5): warning MSB3026: Could not copy "ob
\Program Files\Microsoft Visual Studio\2022\Community\MSBuild\Current\Bin\amd64\Microsoft.Common.CurrentVersion.targets(4632,5): warning MSB3026: Could not copy "ob
\Program Files\Microsoft Visual Studio\2022\Community\MSBuild\Current\Bin\amd64\Microsoft.Common.CurrentVersion.targets(4632,5): warning MSB3026: Could not copy "ob
PRG281Assignmenttwo -> C:\PRG281\PRG281Assignment2\PRG281Assignmenttwo\PRG281Assignmenttwo\bin\Debug\PRG281Assignmenttwo.exe
====== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```



```
            }

            Console.ReadKey();
        }

        1 reference
        public static void Dowork()
        {
            Monitor.Enter(_locked);

                Console.WriteLine($"Thread {Thread.CurrentThread.ManagedThreadId} starting .. ");
                Thread.Sleep(2000);
                Console.WriteLine($"Thread {Thread.CurrentThread.ManagedThreadId} completed .. ");

          Monitor.Exit(_locked);

        }


    }

}
```

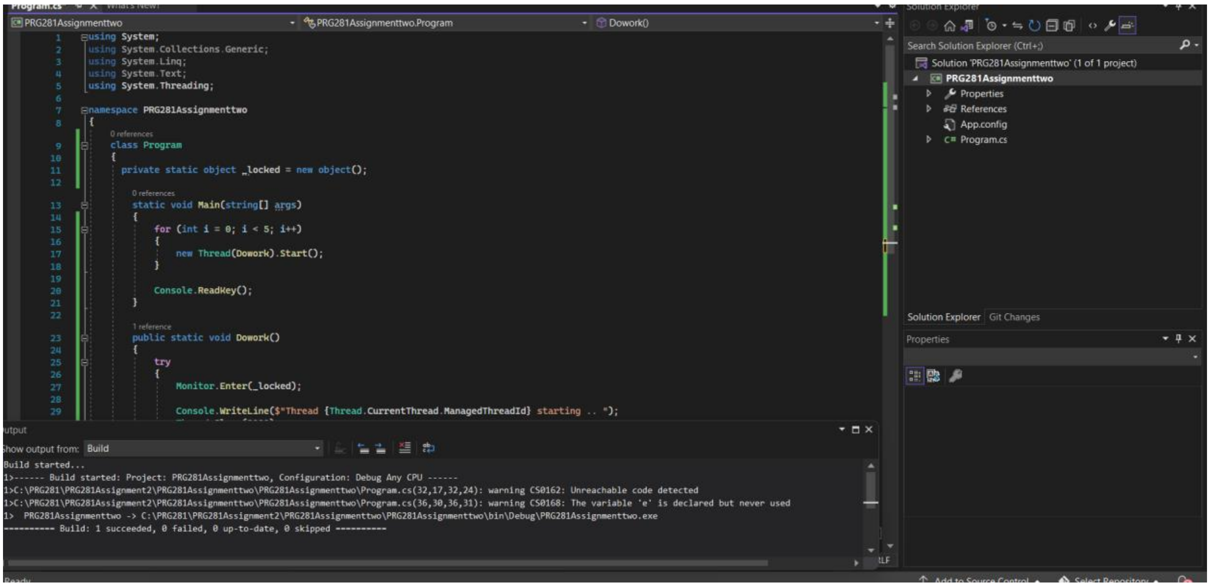
```
C:\Program Files\Microsoft Visual Studio\2022\Community\MSBuild\Current\Bin\amd64\Microsoft.Common.CurrentVersion.targets(4632,5): warning MSB3026: Could not copy "ob
C:\Program Files\Microsoft Visual Studio\2022\Community\MSBuild\Current\Bin\amd64\Microsoft.Common.CurrentVersion.targets(4632,5): warning MSB3026: Could not copy "ob
C:\Program Files\Microsoft Visual Studio\2022\Community\MSBuild\Current\Bin\amd64\Microsoft.Common.CurrentVersion.targets(4632,5): warning MSB3026: Could not copy "ob
C:\Program Files\Microsoft Visual Studio\2022\Community\MSBuild\Current\Bin\amd64\Microsoft.Common.CurrentVersion.targets(4632,5): warning MSB3026: Could not copy "ob
C:\Program Files\Microsoft Visual Studio\2022\Community\MSBuild\Current\Bin\amd64\Microsoft.Common.CurrentVersion.targets(4632,5): warning MSB3026: Could not copy "ob
  PRG281Assignmenttwo -> C:\PRG281\PRG281Assignment2\PRG281Assignmenttwo\PRG281Assignmenttwo\bin\Debug\PRG281Assignmenttwo.exe
========= Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Output screen :



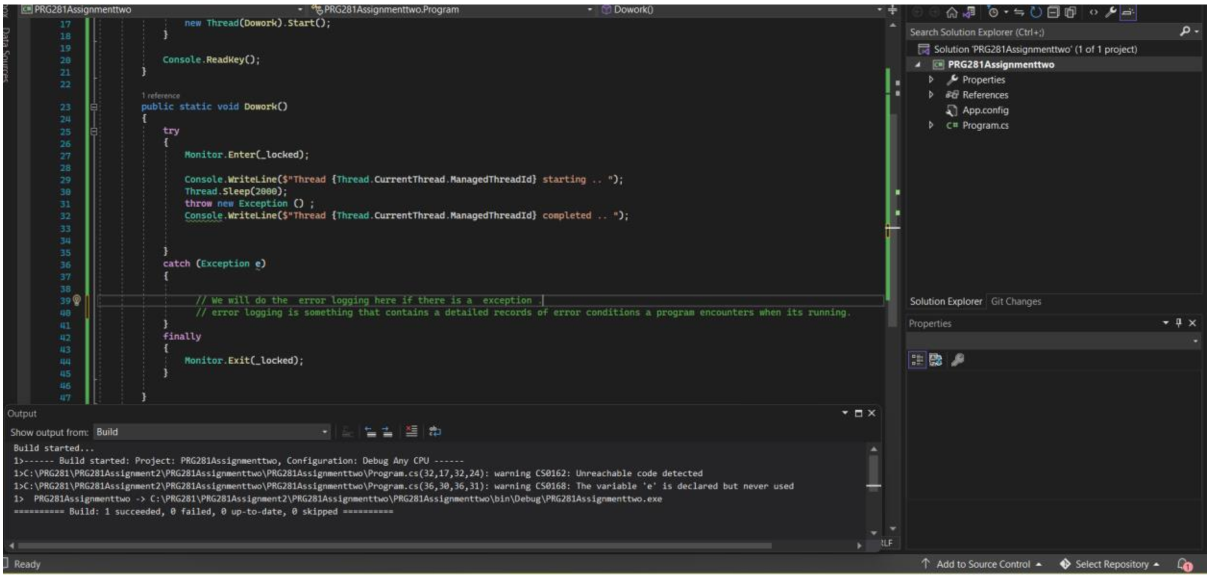
```
Thread 3 starting ..
Thread 3 completed ..
Thread 5 starting ..
Thread 5 completed ..
Thread 4 starting ..
Thread 4 completed ..
Thread 6 starting ..
Thread 6 completed ..
Thread 7 starting ..
Thread 7 completed ..
```

As you can see in the above example we used the same code for the Monitor method , as we used for the lock method , the output screen is the same for both methods but the difference between them is that Monitor you can modify your code in such a way that you can even control your exceptions , meaning for example :



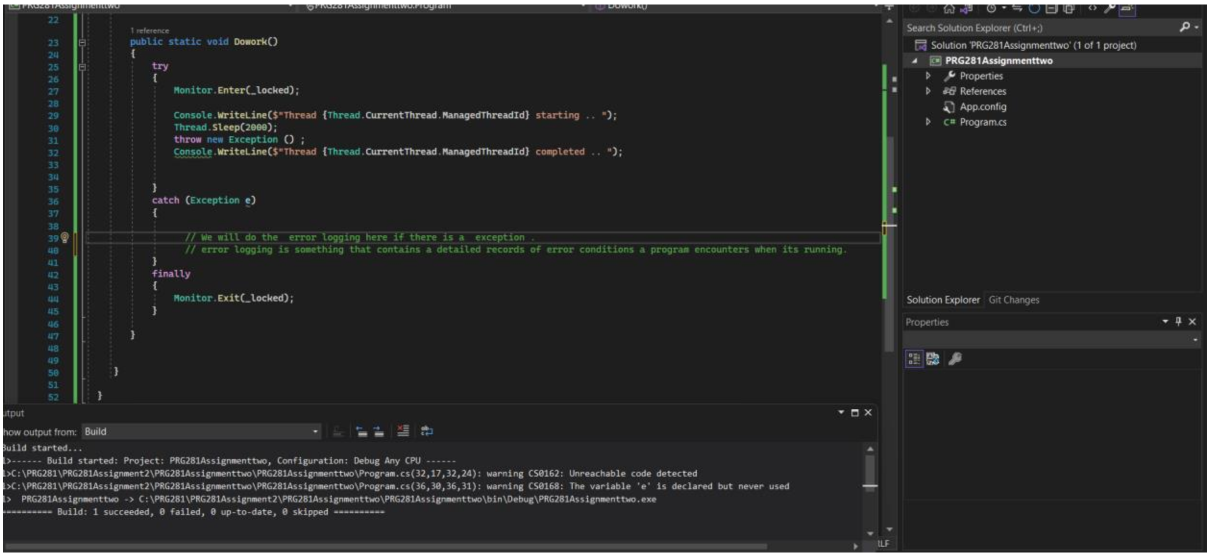
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace PRG281Assignmenttwo
{
    0 references
    class Program
    {
        private static object _locked = new object();

        0 references
        static void Main(string[] args)
        {
            for (int i = 0; i < 5; i++)
            {
                new Thread(Dowork).Start();
            }

            Console.ReadKey();
        }

        1 reference
        public static void Dowork()
        {
            try
            {
                Monitor.Enter(_locked);

                Console.WriteLine($"Thread {Thread.CurrentThread.ManagedThreadId} starting .. ");
```

```
Build started...
1>------ Build started: Project: PRG281Assignmenttwo, Configuration: Debug Any CPU ------
1>C:\PRG281\PRG281Assignment2\PRG281Assignmenttwo\PRG281Assignmenttwo\Program.cs(32,17,32,24): warning CS0162: Unreachable code detected
1>C:\PRG281\PRG281Assignment2\PRG281Assignmenttwo\PRG281Assignmenttwo\Program.cs(36,30,36,31): warning CS0168: The variable 'e' is declared but never used
1>  PRG281Assignmenttwo -> C:\PRG281\PRG281Assignment2\PRG281Assignmenttwo\PRG281Assignmenttwo\bin\Debug\PRG281Assignmenttwo.exe
---------- Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ----------
```



```
                new Thread(Dowork).Start();
            }

            Console.ReadKey();
        }

        1 reference
        public static void Dowork()
        {
            try
            {
                Monitor.Enter(_locked);

                Console.WriteLine($"Thread {Thread.CurrentThread.ManagedThreadId} starting .. ");
                Thread.Sleep(2000);
                throw new Exception () ;
                Console.WriteLine($"Thread {Thread.CurrentThread.ManagedThreadId} completed .. ");


            }
            catch (Exception e)
            {

                  // We will do the  error logging here if there is a  exception .
                  // error logging is something that contains a detailed records of error conditions a program encounters when its running.
            }
            finally
            {
                Monitor.Exit(_locked);
            }

        }
```

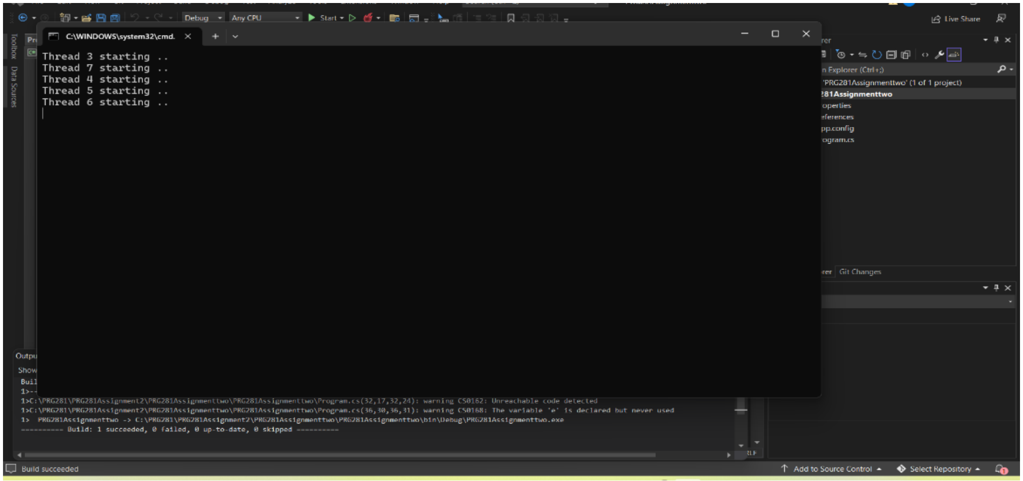
```
Build started...
1>------ Build started: Project: PRG281Assignmenttwo, Configuration: Debug Any CPU ------
1>C:\PRG281\PRG281Assignment2\PRG281Assignmenttwo\PRG281Assignmenttwo\Program.cs(32,17,32,24): warning CS0162: Unreachable code detected
1>C:\PRG281\PRG281Assignment2\PRG281Assignmenttwo\PRG281Assignmenttwo\Program.cs(36,30,36,31): warning CS0168: The variable 'e' is declared but never used
1>  PRG281Assignmenttwo -> C:\PRG281\PRG281Assignment2\PRG281Assignmenttwo\PRG281Assignmenttwo\bin\Debug\PRG281Assignmenttwo.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```



```
22
            1 reference
23          public static void Dowork()
24          {
25              try
26              {
27                  Monitor.Enter(_locked);
28
29                  Console.WriteLine($"Thread {Thread.CurrentThread.ManagedThreadId} starting .. ");
30                  Thread.Sleep(2000);
31                  throw new Exception () ;
32                  Console.WriteLine($"Thread {Thread.CurrentThread.ManagedThreadId} completed .. ");
33
34
35              }
36              catch (Exception e)
37              {
38
39                      // We will do the  error logging here if there is a  exception .
40                      // error logging is something that contains a detailed records of error conditions a program encounters when its running.
41              }
42              finally
43              {
44                  Monitor.Exit(_locked);
45              }
46
47          }
48
49
50      }
51
52  }
```

```
Build started...
1>------ Build started: Project: PRG281Assignmenttwo, Configuration: Debug Any CPU ------
1>C:\PRG281\PRG281Assignment2\PRG281Assignmenttwo\PRG281Assignmenttwo\Program.cs(32,17,32,24): warning CS0162: Unreachable code detected
1>C:\PRG281\PRG281Assignment2\PRG281Assignmenttwo\PRG281Assignmenttwo\Program.cs(36,30,36,31): warning CS0168: The variable 'e' is declared but never used
1>  PRG281Assignmenttwo -> C:\PRG281\PRG281Assignment2\PRG281Assignmenttwo\PRG281Assignmenttwo\bin\Debug\PRG281Assignmenttwo.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Output screen



```
Thread 3 starting ..
Thread 7 starting ..
Thread 4 starting ..
Thread 5 starting ..
Thread 6 starting ..
```

```
1>C:\PRG281\PRG281Assignment2\PRG281Assignmenttwo\PRG281Assignmenttwo\Program.cs(32,17,32,24): warning CS0162: Unreachable code detected
1>C:\PRG281\PRG281Assignment2\PRG281Assignmenttwo\PRG281Assignmenttwo\Program.cs(36,30,36,31): warning CS0168: The variable 'e' is declared but never used
1>  PRG281Assignmenttwo -> C:\PRG281\PRG281Assignment2\PRG281Assignmenttwo\PRG281Assignmenttwo\bin\Debug\PRG281Assignmenttwo.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

As you can see from our current example we have made changes to it , we have added a try catch method , so what the Monitor method will do differently compared to the lock method is that , it will control exceptions , meaning that , since we are starting the thread synchronization at Monitor.Enter(_locked) ; and if somehow if we find some exceptions in the try area code block , we will catch the exception in the catch code block , do the error logging there , and then we will release that log in the finally code block .

As you can see from our edited example now there has been a new exception being thrown at line 31 of the code , basically that will do is that , it throws away the threaded completed statement , so the threaded completed statement will go the catch code block and then finally be released at the finally code block , but the threaded completed statement is not shown in the output screen because it has been thrown with a new exception , but the thread completed statement is still being achieved behind the output screen , but just not showed to the output screen , as the threads are still being synchronized once at a time .

(CODELLIGENT, 2020)

References

CODELLIGENT, 2020. *Youtube.com.* [Online]
Available at: https://www.youtube.com/watch?v=5Zv8fF-KPrE
[Accessed 19 August 2023].

Meikopoulous, O., 2021. *LinkedIn.* [Online]
Available at: https://www.linkedin.com/pulse/c-threading-tasks-async-code-synchronization-part-3-meikopoulos#:~:text=A%20piece%20of%20code%20or,All%20threads%20behave%20properly
[Accessed 20 August 2023].

Oracle, 2023. *Oracle.* [Online]
Available at: https://docs.oracle.com/cd/E13203_01/tuxedo/tux71/html/pgthr5.htm
[Accessed 20 August 2023].