1. Questions 1 (10 points)

1.1 Assume that there is an operating system which provides only Mutex as a synchronization tool. Can you implement a Semaphore using the Mutex? If yes, write a C# code for such a Semaphore class. In no, explain why and justify your answer.

Note: in this question you are requested to implement an integer semaphore (not a binary semaphore).

Yes we Can to implement a Semaphore using the Mutex .

```
namespace Semaphore_
{
    class SemaphoreImpl
    {
        private int current;
        private Mutex mutex;
        private Mutex waiting;

        public SemaphoreImpl( int z , int nThreads)
        {
            current = nThreads;
        }
        public void Signal()
        {
            mutex.WaitOne();
            current += 1;
            if (current > 0)
            {
                mutex.ReleaseMutex();
            }
            if(current >= 0)
            {
                waiting.ReleaseMutex();
            }
        }

        public void WaitOne()
        {
            mutex.WaitOne();
            current -= 1;
            if (current < 0)
            {
                mutex.ReleaseMutex();
                waiting.WaitOne();
            }
            mutex.ReleaseMutex();
        }
    }
}
```

1.2 Recall the Paterson algorithm learnt in the class for two threads. Can you extend the Peterson's algorithm to works with three threads? If yes, show a pseudo code and prove it meets correctly with the three critical sections requirements. If no explain why.

What to submit: pdf file (q1.pdf) with the answers.

We tried to modified the Peterson's algorithm to be :

```
C1 = false
C2 = false
C3 = false
turn = 2

P1{
   while (true){
      C1 = true
      turn = 2
      while ((C2 == true or C3 == true) and (turn == 2 or turn == 3)){
         //spin lock
      }
      //critical section
      C1 = false
   }
}

P2{
   while (true){
      C2 = true
      turn = 3
      while ((C1 == true or C3 == true) and (turn == 1 or turn == 3)){
         //spin lock
      }
      //critical section
      C2 = false
   }
}

P3{
   while (true){
      C3 = true
      turn = 1
      while ((C1 == true or C2 == true) and (turn == 1 or turn == 2)){
         //spin lock
      }
      //critical section
      C3 = false
   }
}
```

But we think that we can't extend the Peterson's algorithm to works with three threads because With more than two threads , the extension will starve the thread without being done, sets turn to the id of a process that will never set turn to any other value (done or terminated) and any other processes neither done or terminated.
In another words, because the logic changes the turn before hitting the spin lock, it seems that a third thread can change the spin lock conditions while another thread is in the critical section, allowing another thread to get in at the same time.