CMPS 311 Homework #4

For this assignment we will complete our dealing with Fibonacci numbers by implementing our own "thread safe" pipe. **The programming of this assignment is to be done *exclusively* on Windows.**

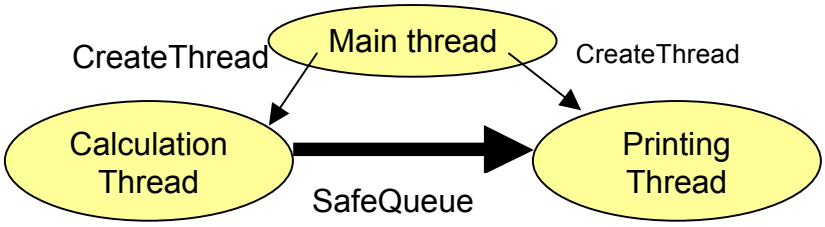
1) As your first task, implement a **class** called SafeQueue in C++ that serves as a FIFO queue with the following functions:

```
class SafeQueue {
public:
    bool isEmpty()          // shall return false if the queue contains
                            // data, true if empty. Should return immediately.

    int read(double & value) /*  shall put the double at the "front" of the queue into
                            "value".  If the queue is empty, an error message should be
                            printed out and a -1 should be return.  Otherwise, return 1. */

     int  write(double value) /*  shall put the "value" onto the back of the queue.  Returns
                                0 if the write failed for any reason, 1 if it succeeded  */
 private:  ...
};
```

You may implement your queue in any reasonable way (linked list, array, etc) - although you may ***not*** use any STL or other library implementations of queues. **15 points**

2) Next, turn your class into a *monitor* so two threads may read and write to the queue w/o race conditions. To do this, you need to include a private mutex variable and acquire the lock (enter the critical section) **within** the read and write methods/functions. Be careful – race conditions and synchronization errors may not always cause visible inconsistencies – this does not mean that the *possibility* for error does not exist. SafeQueue will need to use the CreateMutex, WaitForSingleObject, and ReleaseMutex functions. **(20 points)**

3) Implement problem #1 from Homework #2, this time with threads instead of processes and your SafeQueue class instead of pipes. **(20 points)**



Your main thread should ask the user for an "n" value. It will need to create a SafeQueue and then create the calculation and printing thread. The calculation thread will compute the first "n" Fibonacci numbers, sending each one *individually* to the printing thread via the SafeQueue. The printing thread will read each number from the queue and write them to fib-n.txt. *When you read from the queue, be sure to wait until it is not empty by sitting in a while loop until isEmpty returns false.* The following pseudo code outlines the calculation and printing threads:

Calculation Thread:

```
for ( each Fibonacci number up to "n" numbers)
       safeQueue->write (fibonnaci number);
```

Printing Thread:

```
Open File
For ( each Fibonnaci number up to "n" numbers)
       while ( safeQueue->isEmpty());
              // just sit in tight loop until not empty
       safeQueue->read(num);
       write num to file
Close file
```

4) Enhance your implementation of SafeQueue to eliminate busy waiting. 15 Points

Remove the need to busy wait by adding two new functions Read2 and Write2, along with a separate counting semaphore within your SafeQueue. The value of the semaphore should represent the number of items within the queue. Thus, the Read2 function may efficiently wait (block) until the semaphore is signaled (S>0) before it attempts to enter its critical section and read the first value on the queue.

To do this part, you will need to use the CreateSemaphore, WaitForSingleObject, and ReleaseSemaphore functions. Documentation concerning these functions can be found in your text and in Visual Studio's documentation.

Your enhanced read2 method should follow this general structure:

Wait until Semaphore is > 0
Decrement Semaphore
Enter Critical Section
Read value and remove from queue
Leave Critical Section

Your Enhanced Write2 Function should do the following:

Enter Critical Section
Add value to back of queue
Increment Semaphore
Leave Critical Section

After adding your read2 and write2 methods you will no longer need to use the isEmpty methods – a call to read should block until something has been put onto the queue.

Note, you should use either read / write, or read2/write2 in your main program – if you mix an match, things won't work well!

When you turn in your code, your main program should use read2 and write2. I will check your original read and write functions separately.