1. (a) i. **10 marks** The following precedence graph shows the optimal concurrency possible for a series of statements $S_1..S_7$:

Code the statements using only one COBEGIN and COEND in conjunction with binary semaphores using $P$ and $V$ to achieve the OPTIMAL solution with MINIMAL threads and semaphores. Use pseudo-code for this problem, not $\mu$C++. Use BEGIN and END to make several statements into a single statement and show the initial value (0/1) for all semaphores. Name your semaphores $L_n$, e.g., $L_1, L_2, ...$, to simplify marking.

ii. **3 marks** Draw the process graph for your solution to 1(a)i.

(b) The following is the outline of a correct solution for a bounded buffer using counting semaphores.

```c
uSemaphore full(0), empty(QueueSize);

void Producer::main() {
    for ( ;; ) {
        // produce an item
        empty.P();
        // add element to buffer
        full.V();
    }
    // produce a stopping value
}

void Consumer::main() {
    for ( ;; ) {
        full.P();
        // remove element from buffer
        if (stopping value?) break;
        // process or consume the item
        empty.V();
    }
}
```

i. **2 marks** Does this solution produce maximum concurrency? If not, how can concurrency be increased?

ii. **3 marks** Can this solution handle multiple producers/consumers? If not, explain how to make it safe for multiple producers/consumers. Are there any issues with respect to maximizing concurrency?

(c) **2 marks** With respect to a condition variable, what is a shadow queue? What problem does the shadow queue solve in the readers/writer problem?

(d) **2 marks** What is a private semaphore? What problem does the private semaphore solve in the readers/writer problem?
2. (a) **4 marks** Given this 4-way stop, where the cars are making *no progress*:

![Diagram of 4-way stop]

i. What must have occurred to create this scenario?
ii. What traffic law (rule) attempts to prevent this scenario?
iii. What is the concurrency name for this scenario?
iv. What conditions exist that prevent progress?

(b) Given tasks organized in a ring with a shared resource between each, as in:

```
<table>
<thead>
<tr>
<th>R5</th>
<th>T1</th>
</tr>
</thead>
<tbody>
<tr>
<td>T5</td>
<td>R1</td>
</tr>
<tr>
<td>R4</td>
<td>T2</td>
</tr>
<tr>
<td>T4</td>
<td>R2</td>
</tr>
<tr>
<td>R3</td>
<td>T3</td>
</tr>
</tbody>
</table>
```

and each task needs *both* the resource on the left and right occasionally.

i. **1 mark** Describe a potential for mutual-exclusion deadlock.
ii. **1 mark** Suggest a resource allocation policy (ordering/scheduling) to prevent mutual-exclusion deadlock for this specific scenario.

(c) **4 marks** Does the following resource allocation graph produce a deadlock? Explain your answer by using graph reduction.

![Diagram of resource allocation graph]

(d) **1 mark BONUS:** Who is Sally Sparrow?
3. (a) i. 6 marks Solve the producer and consumer problem with a shared bounded-buffer of integers using the `shared` declaration qualifier and the conditional critical-region, e.g.:

```cpp
shared queue<int,10> s; // example syntax of critical region, not part of solution
region s await s.size() <= 10 {
    // access s
}
```

Write only the declaration of the shared bounded-buffer of size 10 using any appropriate container, and the mutual exclusion and synchronization code using the conditional critical-region.

ii. 1 mark Explain the performance drawback of the conditional critical-region?

(b) 1 mark Explain why a monitor/task destructor must be mutex.

(c) 2 marks How is a condition signal different from a binary semaphore V?

(d) 2 marks A µC++ monitor prevents barging by putting signalled tasks on the acceptor/signaller stack. A Java monitor does not prevent barging, so why does Java need to put signalled tasks somewhere and where does it put them?

(e) 1 mark When monitors are classified using C (calling), W (signalled), and S (signaller) priorities, why are monitors rejected for W < C or S < C?

(f) 2 marks Explain why the following Java barrier is incorrect.

```java
class Barrier {
    // monitor
    private int N, count = 0;
    public Barrier( int N ) { this.N = N; }
    public synchronized void block() {
        count += 1; // count each arriving task
        if ( count < N )
            try { wait(); } catch( InterruptedException e ) {} // barrier full
        else
            notifyAll(); // wake all barrier tasks
        count -= 1; // uncount each leaving task
    }
}
```

4. (a) 2 marks Explain indirect and direct communication.

(b) 7 marks Fill in the 8 entries in the following table with language constructs that provide the corresponding functionality. If there is no language construct, explain why.

<table>
<thead>
<tr>
<th>object properties</th>
<th>member routine properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>thread</td>
<td>stack</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

(c) 2 marks Rewrite the following `_Accept` statement so m2 depends on a different condition, C2, and a different post action, S2.

```c
_when( C1 ) _Accept( m1, m2 ) S1;
```
(d) **2 marks** Explain why an administrator must use `signalBlock` rather than `signal` to unblock worker tasks.

(e) **1 mark** f is a \( \mu \)C++ future. Why can the following accesses fail?

```cpp
osacquire( cout ) << f() << ’ ’ << f + 1 << end;
```

(f) **2 marks** Given the following statement:

```cpp
_Select( f1 ) osacquire( cout ) << ”f1” << endl;
or _Select( f2 ) osacquire( cout ) << ”f2” << endl;
and _Select( f3 ) osacquire( cout ) << ”f3” << endl;
```

and the futures become available in the order f3, f1, and f2, what is the output?

5. (a) **1 mark** Why do compilers and hardware reorder statements?

(b) **2 marks** What is a *cache line* and what problem can it introduce in a concurrent program?

(c) **2 marks** What is the purpose of the `volatile` qualifier in C/C++ with respect to safe variable access in a concurrent program?

(d) **2 marks** When using the LL/SC instructions to build lock-free data structures, why is there no ABA problem?

(e) **2 marks** Why is programming an *if* statement difficult in GPU programming?

(f) **2 marks** Java concurrency design has task start and join members. Discuss the effects of this design.

6. (a) **14 marks** In the *group mutual-exclusion* problem, each task is assigned a *session*. Sessions represent a kind of access to a resource. Tasks with the same session may use the resource simultaneously, but different sessions may not use the resource simultaneously. For example, if session 3 tasks are using the resource, session 2 and 5 tasks must wait.

   Implement group mutual-exclusion using a \( \mu \)C++ monitor with internal scheduling and the following interface:

   ```cpp
   _Monitor GME {
   public:
   void start( unsigned int session ); // start using resource
   void end(); // finish using resource
   }
   ```

   Session tasks must be service in temporal (FIFO) order, and multiple tasks with the same session number must be services simultaneously. Write only the GME monitor, **do not write the session tasks or program main**.

   (b) **3 marks** Can the group mutual-exclusion problem be written using external scheduling? If not, give all reasons why not.

7. **23 marks** Write an administrator task for the Celebrities restaurant. The Celebrities administrator is the *host* assigning diners to tables. To maximize table use (and hence the generation of income), diners are grouped into tables of size `TABLE_SIZE` and assigned a waiter. Hence, people sit at the table where they are assigned with the people selected by the host.

   The host creates `NUM_WAITERS` waiters, and each diner who makes a reservation with the host is immediately assigned a (future) waiter, who eventually greets them on arrival and takes them to a table.
When *celebrities* come to the restaurant, the host wants to improve publicity by giving them an enhanced experience. Thus, a celebrity skips the lineup (barges) and is assigned the first free waiter and dines alone.

When a table is emptied, the associated waiter returns to the host, gives the host any tip from the table, and waits for another table assignment. (A waiter arrives initially with a 0 tip.)

When the restaurant closes for the night (i.e., the *Host’s* destructor is called), the host waits for all the waiters to finish their tables, and prints the amount of each waiter’s tip, which is the total amount of the collected tips divided by the number of waiters. Assume there are no diners waiting for tables when the restaurant closes (this implies that the total number of regular diners is a multiple of *TABLE_SIZE*).

The *Host* has the following interface (you may add only a public destructor and private members):

```cpp
class Host {
    public:
        class Waiter {
            ...
            public:
                Waiter( Host &host ) : host( host ) {};
        };
        typedef Future<ISM<Waiter>> FWaiter; // future waiter
        _Event Closed {}; // indicate restaurant closed
    private:
        const int NUM_WAITERS, TABLE_SIZE;
        list<FWaiter> diners, celebrities;
        uCondition waiterBench;
        unsigned int tips = 0;
        bool closed = false;
    public:
        Host( const int NUM_WAITERS, const int TABLE_SIZE ) :
            NUM_WAITERS( NUM_WAITERS ), TABLE_SIZE( TABLE_SIZE ) {};
        void requestWork( int tip ) {
            tips += tip;
            waiterBench.wait( (uintptr_t)&uThisTask() );
            if ( closed ) _Throw Closed();
        }
        FWaiter reserveTable( bool isCelebrity ) { // diner given future id of waiter
            FWaiter fw;
            if ( isCelebrity ) {
                celebrities.push_back( fw ); // store future
            } else {
                diners.push_back( fw ); // store future
            }
            return fw;
        }
    private:
        // YOU MAY ADD OTHER DATA AND ROUTINES
        void main(); // YOU WRITE THIS ROUTINE
};
```

Ensure the administrator task does as much administration works as possible; a monitor-style solution will receive little or no marks. Write the code for Host::main and any necessary declarations/initializations; do NOT write the diner, celebrity, waiter or program main. Assume the program main creates and deletes all the necessary tasks, appropriately.
\( \mu \text{C++} \) future server operations are:

- \texttt{delivery(T \ \text{result})} – copy result to be returned to the client(s) into the future, unblocking clients waiting for the result.
- \texttt{exception(uBaseEvent \cdot \text{cause})} – copy a server-generated exception into the future, and the exception \text{cause} is thrown at clients accessing the future.

\text{C++} list operations are:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{int \ size()}</td>
<td>list size</td>
</tr>
<tr>
<td>\texttt{bool \ empty()}</td>
<td>size() == 0</td>
</tr>
<tr>
<td>\texttt{T &amp; \ front()}</td>
<td>first element</td>
</tr>
<tr>
<td>\texttt{T &amp; \ back()}</td>
<td>last element</td>
</tr>
<tr>
<td>\texttt{void \ push_front(const \ T \ &amp; \ x)}</td>
<td>add ( x ) before first element</td>
</tr>
<tr>
<td>\texttt{void \ push_back(const \ T \ &amp; \ x)}</td>
<td>add ( x ) after last element</td>
</tr>
<tr>
<td>\texttt{void \ pop_front()}</td>
<td>remove first element</td>
</tr>
<tr>
<td>\texttt{void \ pop_back()}</td>
<td>remove last element</td>
</tr>
<tr>
<td>\texttt{void \ clear()}</td>
<td>erase all elements</td>
</tr>
</tbody>
</table>