✓ **Congratulations! You passed!**

TO PASS 70% or higher

Keep Learning

GRADE
72.5%

Module 1 Quiz

LATEST SUBMISSION GRADE

72.5%

1. **Which of the following primitive operations on threads were introduced in Lecture 1.1?** — 0.75 / 1 point

   Please choose all options that are correct.

   ! Incorrect

2. **Which of the following statements are true, based on what you learned about threads in Lecture 1.1?** — 0.5 / 1 point

   Please choose all options that are correct.

   ! Incorrect

   Please choose all options that are correct.

   ! Incorrect

4. **Consider a parallel program with two structured locks, L1 and L2, (as introduced in Lecture 1.2), and 100 threads that are started in parallel such that each of them executes the following snippet of code:** — 1 / 1 point

```
synchronized(L1) {
  synchronized(L2) {
    S1
  }
}
```

   The statement "This program is guaranteed to be free from deadlocks" is:

   ✓ Correct

5. **Which of the following are true regarding unstructured read-write locks introduced in Lecture 1.3?** — 0.667 / 1 point

   Please choose all options that are correct.

6. **Which of the following best describe the differences between structured and unstructured locks introduced in Lectures 1.2 and 1.3?** — 0.667 / 1 point

   Please choose all options that are correct.

   ! Incorrect

7. **Please classify the following five scenarios, each employing two threads (T1 and T2) running concurrently, as possibly leading to deadlock, livelock or neither. Recall that deadlock and livelock were discussed in Lecture 1.4.** — 0 / 1 point

   Scenario 1:

```
// Thread T1
{
  . . . // sequential code
  T2.join();
}

// Thread T2
{
  . . . // sequential code
  T1.join();
}
```

   Scenario 2: **A and B are declared and initialized as distinct objects**

```
// Thread T1
{
  synchronized(A) {
    synchronized(B) {
      . . . // sequential code
```

```
  synchronized(A) {
    synchronized(B) {
      . . . // sequential code
    }
  }
}
```

   Scenario 3: **A and B are declared and initialized as distinct objects**

```
// Thread T1
{
  synchronized(A) {
    synchronized(B) {
      . . . // sequential code
    }
  }
}

// Thread T2
{
  synchronized(B) {
    synchronized(A) {
      . . . // sequential code
    }
  }
}
```

   Scenario 4: **x is declared and initialized as a shared Integer object**

```
// Thread T1
{
  while (x > 4) {
    synchronized(x) {
      x--;
    }
  }
}

// Thread T2
{
```

   Scenario 5: **x is declared and initialized as a shared Integer object**

```
// Thread T1
{
  while (x < 6) {
    synchronized(x) {
      x++;
    }
  }
}

// Thread T2
{
  while (x < 8) {
    synchronized(x) {
      x++;
    }
  }
}
```

   ! Incorrect

8. **What is the difference between deadlock and livelock? (Recall that deadlock and livelock were discussed in Lecture 1.4.)** — 1 / 1 point

9. **Many algorithms have been proposed to address the Dining Philosophers problem introduced in Lecture 1.5. In this problem, you will evaluate one such algorithm.** — 1 / 1 point

   The following simple algorithm can deadlock with all philosophers holding their left chopsticks.

```
// All philosophers:
while (True) {
  acquire philosopher's left chopstick
  acquire philosopher's right chopstick
  eat
  release right chopstick
```

   So, we develop a slight variant that attempts to avoid this problem — adjacent philosophers pick up the chopsticks in the opposite order. More precisely, we propose separate algorithms for the even-numbered and odd-numbered philosophers. Assume the philosophers are numbered 1, ..., n clockwise around the table.

```
// All even-numbered philosophers:
while (True) {
  acquire philosopher's left chopstick
  acquire philosopher's right chopstick
  eat
  release right chopstick
  release left chopstick
}

// All odd-numbered philosophers:
while (True) {
  acquire philosopher's right chopstick
  acquire philosopher's left chopstick
  eat
  release left chopstick
  release right chopstick
}
```

   What liveness issues could arise if we have an even number of philosophers? What about if we had an odd number of philosophers?

   ✓ Correct

10. **Towards the end of Lecture 1.5, Dr. Sarkar mentioned that deadlock and livelock can be avoided in the Dining Philosophers problem by modifying the "all acquire left fork first" algorithm such that n-1 philosophers attempt to acquire their left fork first, and 1 philosopher attempts to acquire its right fork first. However, nothing was mentioned about the impact of this modification on another important liveness issue: starvation. How could starvation occur, or not occur, with this modification presented at the end of Lecture 1.5?** — 1 / 1 point