1. Example to illustrate interleavings: say that thread A executes \( f() \) and thread B executes \( g() \). (Here, we are using the term "thread" abstractly. This example applies to any of the approaches that fall under the word "thread".)

   a. [this is pseudocode]

   ```
   int x;
   int main(int argc, char** argv) {
       tid tid1 = thread_create(f, NULL);
       tid tid2 = thread_create(g, NULL);
       thread_join(tid1);
       thread_join(tid2);
       printf("%d\n", x);
   }
   ```

   b. What are possible values of \( x \) after A has executed \( f() \) and B has executed \( g() \)? In other words, what are possible outputs of the program above?

   ```
   void f() {
       x = 1;
       thread_exit();
   }
   ```

   ```
   void g() {
       x = 2;
       thread_exit();
   }
   ```

   What are possible values of \( x \) after A has executed \( f() \) and B has executed \( g() \)?

   c. What are possible values of \( x \)?

2. Linked list example

   ```
   struct List_elem {
       int data;
       struct List_elem* next;
   }
   ```

   ```
   List_elem* head = 0;
   ```

   ```
   insert(int data) {
       List_elem* l = new List_elem;
       l->data = data;
       l->next = head;
       head = l;
   }
   ```

   What happens if two threads execute `insert()` at once and we get the following interleaving?

   ```
   thread 1: l->next = head
   thread 2: l->next = head
   thread 2: head = l;
   thread 1: head = l;
   ```
3. Producer/consumer example:

```c
/*
 * "buffer" stores BUFFER_SIZE items
 * "count" is number of used slots. a variable that lives in memory
 * "out" is next empty buffer slot to fill (if any)
 * "in" is oldest filled slot to consume (if any)
 */

void producer (void *ignored) {
    for (;;) {
        /* next line produces an item and puts it in nextProduced */
        nextProduced = means_of_production();
        while (count == BUFFER_SIZE)
            ; // do nothing
        buffer[in] = nextProduced;
        in = (in + 1) % BUFFER_SIZE;
        count++;
    }
}

void consumer (void *ignored) {
    for (;;) {
        while (count == 0)
            ; // do nothing
        nextConsumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;
        /* next line abstractly consumes the item */
        consume_item(nextConsumed);
    }
}
```

4. Some other examples. What is the point of these?


a. Can both "critical sections" run?

```c
int flag1 = 0, flag2 = 0;

int main () {
    tid id = thread_create (p1, NULL);
    p2 (); thread_join (id);
}
```

```c
void p1 (void *ignored) {
    flag1 = 1;
    if (!flag2) {
        critical_section_1 ();
    }
}
```

```c
void p2 (void *ignored) {
    flag2 = 1;
    if (!flag1) {
        critical_section_2 ();
    }
}
```

b. Can use() be called with value 0, if p2 and p1 run concurrently?

```c
int data = 0, ready = 0;

void p1 () {
    data = 2000;
    ready = 1;
}
```

```c
void p2 () {
    while (!ready) {}
    use(data);
}
```

```c
int a = 0, b = 0;

void p1 (void *ignored) {
    a = 1;
}
```

```c
void p2 (void *ignored) {
    if (a == 1)
        b = 1;
}
```

```c
void p3 (void *ignored) {
    if (b == 1)
        use (a);
}
```