CSP2348/CSP5243 Data Structures

Review Questions 10

Topic: Queue and List ADTs

1. Would it make sense to call a queue a LILO (last-in-last-out) structure?

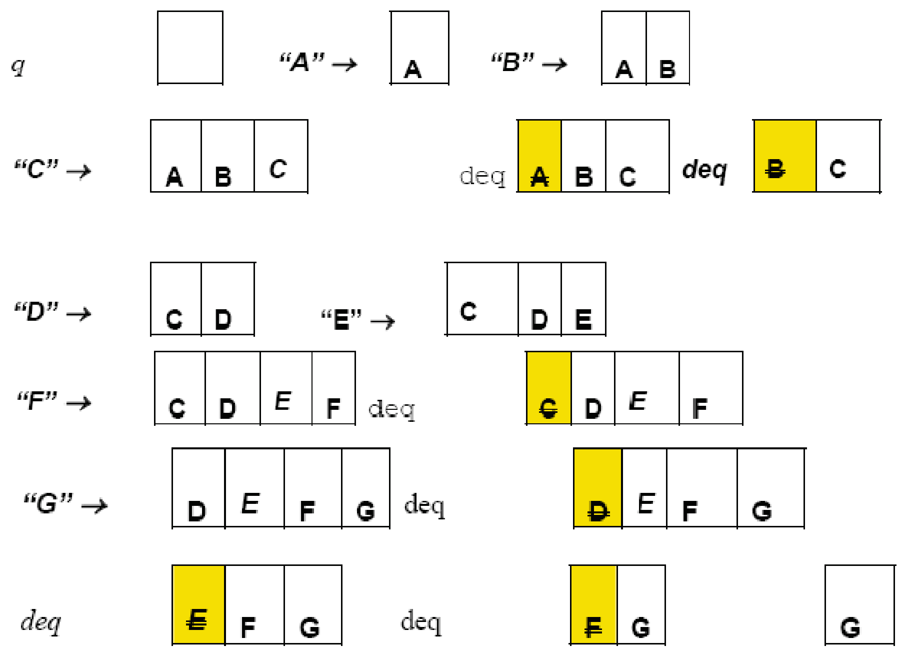
   Yes. **LILO** = **FIFO**

2. Attempt Exercises 7.4 on page 167 in the textbook.

   It would be pointless to implement the queue ADT using a DLL, since none of the operations needs to access any node's predecessor.

3. Trace the following code, showing the contents of the queue q after each call [note: `enqueue()`= `addLast(); dequeue()` = `removeFirst()`]

```
ArrayQueue q;
q.enqueue("A");
q.enqueue("B");
q.enqueue("C");
q.dequeue();
q.dequeue();
q.enqueue("D");
q.enqueue("E");
q.enqueue("F");
q.dequeue();
q.enqueue("G");
q.dequeue();
q.dequeue();
q.dequeue();
```



4. Explain whether the expression is true or false:

   Feeling = <<I, want, to, pass, this, exam, but, I, do, not, know, if, I, can, pass, it>>

   It's a list because list allows existence of replicated values.

5. In deciding whether to use an `ArrayList` or a `LinkedList` in an application, what factors make one choice better than the other?

   An `ArrayList` **object should be preferred when frequent lookups are expected. A** `LinkedList` **object should be preferred when frequent additions and/or removals are expected.**

6. Attempt Exercises 8.3 on page 199 in the textbook.

   Using the `List` **ADT of Program 8.2, a possible version of the** `reorder` **method is as follows:**

```
static List reorder (List persons) {
// Assume that persons is a list of Person objects, ordered by
name.
// Return a similar list of Person objects, ordered such that
all
// children (aged under 18) come before all adults (aged 18 or
// over), but otherwise preserving the ordering by name.
      List children = new LinkedList();
      List adults = new LinkedList();
      Iterator iter = persons.iterator();
      while (iter.hasNext()) {
            Person p = (Person) iter.next();
            if (p.age <= 18)
                  children.add(p);
            else
                  adults.add(p);
      }
      // Construct the result with children before adults.
      List result = children;
      result.addAll(adults);
      return result;
}
```