CSc 127B — Introduction to Computer Science II
Fall 2015 (McCann)

Answers to the Practice Questions ahead of Exam #2

The section leaders were kind enough to write some practice questions for you to work as you prepare for the exam; those are in another handout. These are the SLs' suggested answers for the questions.

1.

```
for(int i = occupancy; i >0; i--){
    letterArray[i] = letterArray[i-1];
}
letterArray[0] = 'b';
occupancy++;
letterArray[occupancy] = 't';
```

2.

```
char[] newArray = new char[letterArray.length-1];
for(int i = 0; i < newArray.length; i++){
    newArray[i] = letterArray[i+1];
}
occupancy--;
```

3.

```
if(capacity == occupancy){
    char[] newArray = new char[2*capacity];
    for(int i = 0; i < letterArray.length; i++){
        newArray[i] = letterArray[i];
    }
    newArray[occupancy++] = z;
    letterArray = newArray;
} else {
    letterArray[occupancy++] = z;
}
```

4.

```
public static int log(int base, int result)
{
    if (result == 0) return 0;

    return 1 + log(base, result / base);
}
```

(Continued . . .)

5.

```java
public static void partition(String word)
{
    if (word.length() == 0) return;

    partitionHelper(word);
    partition(word.substring(1));
}

public static void partitionHelper(String word){
    if (word.length() == 0) return;

    System.out.println(word);
    partitionHelper(word.substring(0,word.length() - 1));
}
```

6.

```java
public static String commonPrefix(String first, String second)
{
    if (first.equals("") || second.equals("")) return "";

    char firstsChar = first.charAt(0);
    char secondsChar = second.charAt(0);

    if( firstsChar != secondsChar ) return "";

    return firstsChar + commonPrefix(first.substring(1),second.substring(1));
}
```

7.

```java
public static boolean prime(int n)
{
    return primeHelper(n, 2);
}

public static boolean primeHelper(int n, int divisor)
{
    if (divisor == n) return true;
    if (n % divisor == 0) return false;

    return primeHelper(n, divisor + 1);
}
```

(Continued . . .)

8. (a)

```
          +---+---+    +---+---+    +---+---+
head --> | A | --+--> | B | --+--> | D | / +
          +---+---+    +---+---+    +---+---+
                         ^
                         |
                      currNode
```

(b)

```
class Node{
    public char data;
    public Node next;
}
```

(c) currNode's next field gets the reference held by the next field of the node following currNode.

(d)

```
public void removeNode(Node currNode) {
    currNode.next = currNode.next.next;
}
```

(e) No. You cant remove the head of the list, because currNode can't reference a node ahead of the first node.

(f)

```
public void removeNode(Node currNode) {
    if (currNode == null) {
        head = head.next;
    } else {
        currNode.next = currNode.next.next;
    }
}
```

(g)

```
public void remove(char c) {
    if (head == null) return;  // Nothing to delete
    if (head.data == c) removeNode(null); // delete first node
    Node temp = head;
    while (temp.next != null && temp.next.data != c) {
        temp = temp.next;
    }
    if (temp.next == null) return;  // didn't find the data
    removeNode(temp);  // did find it, so need to remove its node
}
```

9.

```
     (a)                                          (b)

|  Once     |                               |  Once     |
|  Upon     |                               |  Once     |
|  A        |                               |  Upon     |
|  Midnight |                               |  Upon     |
|  Dreary   |                               |  A        |
+-----------+                               |  A        |
                                            |  Midnight |
                                            |  Midnight |
                                            |  Dreary   |
                                            |  Dreary   |
                                            +-----------+
```

10. Pop from each of the two original stacks and push the largest of the two values onto a single stack. Repeat until both of the original stacks are empty. Now pop elements off this stack and push them onto the final stack (this ensures that the original order is maintained rather than reversed).

11. ArrayLists are preferable because we can instantaneously access their last element (stack.size() - 1) whereas LinkedLists require a node by node traversal to get to that element.

12. (a) Prints the first n Fibonacci numbers
    (b) A line at an amusement park
    (c)
        i. first, first
        ii. last, first
    (d) When using a circular array to implement a Queue, the empty and full condition using the front and rear pointers are both the same. This means if we used front and rear we would not be able to tell the difference between full and empty.
    (e) It doesn't exist. There is only a Queue interface, and ArrayDeque as one implmentation.

13. (a)

```
         +---+----+---+    +---+----+---+    +---+----+---+    +---+----+---+
         |   |    |   | <--+-- |    |   | <--+-- |    |   | <--+-- |    |   |
head --> | / | 01 |   |    |   | 02 |   |    |   | 03 |   |    |   | 04 | / | <-- tail
         |   |    | --+--> |   |    | --+--> |   |    | --+--> |   |    |   |
         +---+----+---+    +---+----+---+    +---+----+---+    +---+----+---+
```

    (b)

```
public Node reverseDouble(Node head)
{
    if (head == null)  return null ;
    Node temp=head;
    head=tail;
    tail=temp;
    temp2=head;
    while(temp2!=null) {
        temp=temp2.next;
        temp2.next=temp2.prev;
        temp2.prev=temp;
        temp2=temp2.next;
    }
    return temp2;
}
```

    (c) The previous reference for the next node, the next reference for the previous node, and setting the references for the node being inserted.

14.

```
public boolean isCircular(Node<Integer> head)
{
    Node<Integer> temp = head;
    while (temp != null) {
        temp = temp.getNext()
        if (temp == head) return true;
    }
    return false;
}
```