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

Practice Questions ahead of Exam #2

The section leaders were again kind enough to write some practice questions for you to work as you prepare for the exam. We'll supply answers in a few days. These questions will be the most valuable if you first study, then sit down and answer the questions in an exam-like setting (quiet location, no computer). If you look at the answers first, you are more likely to say, "Oh, I knew that," even if you really didn't.

1. Earlier this semester we learned how to "grow" our own arrays once we couldnt add any more elements to our current array. Let's look at the example array below:

```
  0   1   2   3   4   5
+---+---+---+---+---+---+
| a | c | r | g |   |   |
+---+---+---+---+---+---+
```

   We can see that our array has 4 elements in it with room for more elements. We can still add items to our array-based list. Write the necessary code to first prepend the char 'b' to the front of our list and to then append the char 't' to the end of our list.

   Assume the following:

   - The array reference variable to our array of characters (char) is `letterArray`.
   - You have the occupancy (the number of items currently stored in the array) stored in a variable called occupancy.
   - The arrays current state is the one pictured/described above.
   - The array is an array of type char.

2. Using the same starting array as the previous question's, write the code necessary for deleting the first element (and by doing so shortening the capacity of our list down by 1 (from 6 to 5).

3. Let's say we have the array where we dont know the whether we have to grow the array or not and we wish to append our array. Write the code to first check for whether we must grow our array, and if so "grow" the array. Then append the character 'z' to the end of our list. Make sure you write the code to check whether we need to grow the array before you actually grow the array. Do not forget to add the item to the end of the list (whether you grow the array or not). Grow the arrays size to double its previous size.

   Assume the following:

   - The array reference variable to our array of characters (char) is `letterArray`.
   - You have the occupancy (the number of items currently stored in the array) stored in a variable called occupancy.
   - You have the capacity of the array stored in a variable called capacity.
   - The arrays current state is the one pictured/described above.
   - The array is an array of type char.

(Continued . . .)

4. Recursively finding the logarithm of a value, knowing its base.

   So, weve seen before that exponentiation can be viewed as repeated multiplication. For example, 10^3 can be viewed as $10 * 10 * 10 = 1000$. A logarithm function is used to find the power we need to take a number to to get a given result. So, for instance $log_{10}1000 = 3$. This is saying that we would have to take the base (10) to the third power in order to get 1000. Just as we were able to find the result to an exponential function using recursion, we can also do the same for the logarithm. Its as simple as undoing the multiplication. (Now, which operation undoes multiplication?)

   Signature to help you out: `public static int log(int base, int result)`

   Here are some example problems that your code should be able to solve:

   - 10n = 1000; Answer should be 3
   - 5n = 390625; Answer should be 8
   - 8n = 1152921504606846976; Answer should be 20

   Note: Only try to solve this on integer powers. N must be an integer or else things get weird. And you have to start using the power series and thats lame. Integers are the only real numbers to me.

5. Partitioning a group.

   Given a string, recursively print all substrings within that maintain order.

   Hint: You should view this as finding all prefixes of a word, and then finding recursing on the word minus the first letter. So, you could find Post by considering "Post", "Pos," "Po", "P"], then do the same for "ost", then "st", then "t". Then youve found all substrings.

   Example: "Post" contains the following ordered substrings: "Post","Pos", "Po", "P", "ost", "os", "o", "st", "s", and "t".

   In general you will print out $\frac{n(n+1)}{2}$ substrings. You may recognize this as the sum of all numbers between 1 and n. Weird that it works like that. Consider breaking this up into two functions: One that finds all prefixes of a string, and one that finds all suffixes of a string.

   This one is pretty hard, dont get discouraged if you it is taking awhile to figure out.

6. Given two strings, recursively find the largest common prefix amongst the strings.

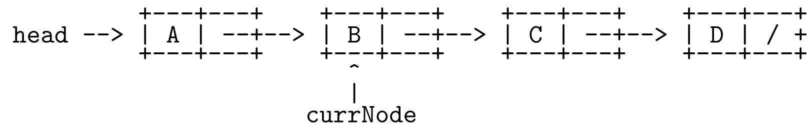
   For instance, we have a "Post" and "Postal", the largest common prefix of this is "Post". If we had "That" and "Other", the largest common prefix would be the empty string. Given "Alone" and "Alone", the largest prefix would be "Alone". Given "WhateverIWant" and the empty string, the largest prefix would be the empty string.

7. Write a method prime(int n) that recursively discovers if n is prime. Prime means that a number is only divisible by 1 and itself, with no remainder. Moving the recursion to a helper method is useful here.

(Continued . . .)

8. (Singly) Linked-Lists.

Consider the drawing of the linked list shown below. currNode points to an element that we would like to remove from our list. Each of the nodes have both a data field (a char) and a reference to the next node.



(a) Draw a picture to reflect what would happen if we delete the node after the one that currNode points to.

(b) Write a simplified node class (no methods, no generics) that includes the two attributes of each node. Make each public.

(c) Next, explain (in words, not code) how to remove the node located after currNode from the list.

(d) Write a method named `removeNode()` that accepts a reference to the node ahead of the node to be removed, and removes it from the list. In other words, turn your answer to the last question into code.

(e) Is your answer to the previous question able to delete any node in a linked list?

(f) Rewrite your code for `removeNode()` so that it deletes any node in the list. If the argument is null, the first node of the list is to be deleted.

(g) Last but not least, write a method called `remove()` that accepts a data value and uses your `removeNode()` method in order to remove the node closest to the front of the list that is holding that data value.

(Continued ...)

9. Write your answers for comments (a) and (b) in the spaces provided below the following code:

```
1   import java.util.Stack;
2
3   public class StackQuestion1
4   {
5
6       public static void main(String[] args)
7       {
8           Stack<String> s = new Stack<String>();
9           s.push("Dreary");
10          s.push("Midnight");
11          s.push("A");
12          s.push("Upon");
13          s.push("Once");
14          s.peek();
15
16          /*** (a) Draw s at this point ***/
17
18          Stack<String> temp = new Stack<String>();
19          int i = 0;
20
21          while (!s.isEmpty())
22          {
23              switch (i % 2)
24              {
25              case 0:
26                  temp.push(s.peek());
27                  i++;
28                  break;
29              case 1:
30                  temp.push(s.pop());
31                  i++;
32                  break;
33              default:
34                  break;
35              }
36          }
37
38          while (!temp.isEmpty())
39              s.push(temp.pop());
40
41          /*** (b) Draw s at this point ***/
42
43      }
44
45  }
```

(a) (b)

(Continued . . .)

10. If I give you two stacks of integers in sorted order (largest at the top, smallest at the bottom), explain how you would combine them into a single stack that has the same order. You may use as many extra stacks as you want, but only stacks may be used.

    For example, if the context of Stack1 is: (Bottom) 3 10 25 (Top) and of Stack2 is: (B) 1 9 17 56 (T), the resulting stack's content needs to be: (B) 1 3 9 10 17 25 56 (T).

    Note: Just an explanation is sufficient. No code or pseudo-code is necessary.

11. Weve used `get(stack.size() - 1)` and `remove(stack.size() - 1)` to implement peek() and pop() in our own stack classes using lists as our underlying data structure. Both ArrayList and LinkedList have these methods, but why might we want to use one of those two classes over the other?

12. Read and evaluate the following function that uses a Queue of integers:

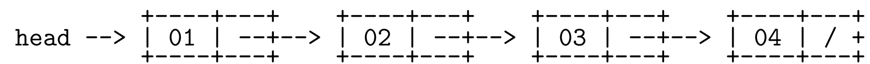
```
1  void method(int n)
2  {
3      Queue<Integer> q = new Queue<Integer>();
4      q.enqueue(0);
5      q.enqueue(1);
6      for (int i = 0; i < n; i++){
7          int a = q.dequeue();
8          int b = q.dequeue();
9          q.enqueue(b);
10         q.enqueue(a + b);
11         System.out.println(a);
12     }
13 }
```

    (a) What does this function do?
        i. Prints the first n Fibonacci numbers
        ii. Prints the integers from 0 to n - 1
        iii. Prints the odd integers from 0 to n
        iv. Prints the integers from n - 1 to 0
    (b) A lot of metaphors get used in computer science, a common one for explaining queues is:
        i. A farmer copying himself to harvest wheat faster.
        ii. A box, with a pointer to the next box
        iii. A stack of plates
        iv. A line at an amusement park
    (c) Fill in the blanks with first or last:
        i. Queues are considered __________ in, __________ out.
        ii. Stacks are considered __________ in, __________ out.
    (d) When using a circular array to implement a Queue we often maintain an occupancy variable to determine isFull() and isEmpty() instead of front and rear pointers. Why?
    (e) What is important to know about Javas Queue class?

(Continued . . .)

13. Doubly-Linked Lists.

   (a) Redraw the singly linked list shown below as a doubly linked list.



   (b) Implement a method that reverses a doubly linked list.

   The method signature should be as follows:

```
Node<Integer> reverseDouble(Node<Integer> head)
```

   head will point to a doubly linked list and the method should return the head of the newly converted doubly linked list.

   (c) What references do you have to take into consideration when inserting into a doubly linked list?

14. Implement a function to check and see if a singly linked list is circular or not.

   The method signature is as follows:

```
boolean isCircular(Node<Integer> head)
```

   The method should return true if the list passed is circular and false if the list is not. The head will always point to the first node of the list.