CSc 127B Exam 2 Practice Test

The best way to use this exam to study is to try and take the test once without any notes. Then, use the questions you struggled with or got wrong to decide what to study. However, please note that, while the SLs do their best to write questions that will reflect the test, there may be some things that are not on this practice test that appear on the real test. So, make sure to study your notes and the lectures in addition to taking this!

1. Fill in the blanks for the following

a) In a stack, the first thing put on the stack is the _____last_____ thing taken out.

b) The three basic functions of a stack are: ___push______, ____pop______ and ____peek______.

c) In a queue, the first thing put in the queue is the ____first_____ thing taken out.

d) The two basic functions of a queue are: ___queue______ and ____dequeue______

e) A queue which partially sorts the data enqueued based on its importance is called a _________priority queue______

2. We learned a bit about stacks and queues in class. Below is the output of a mystery class after various functions have finished executing. PutOn will put an int onto the stack or queue and TakeOff will take the int off the stack or queue. Based on the toString is the mystery class a stack or a queue?

Input:
PutOn(7); putOn(10); putOn(42); takeOff(); takeOff(); putOn(7);
putOn(7); System.out.println(toString());
Output: 7 7 7

Answer: Stack!

2) What is a potential error you would want to check for when using this class?

Answer: Trying to takeOff from an empty stack.
3) Using a `ArrayList`, finish this unknown class.

```java
class Unknown {
    public ArrayList<Integer> arrayList = new ArrayList<>();
    public Unknown() {
    }
    // Write putOn
    // Write takeOff
    public String toString() {
        StringBuilder str = new StringBuilder();
        for (int i = 0; i < arrayList.size(); i++) {
            str.append(arrayList.get(i) + " ");
        }
        return str.toString();
    }
}
```

Answer: There are other ways to do it, but this is the first I thought of:

```java
public void putOn(int toPut) {
    arrayList.add(toPut);
}

public int takeOff() {
    if (arrayList.size() == 0) {
        return null;
    }
    int temp = arrayList.get(arrayList.size()-1);
    arrayList.remove(arrayList.size()-1);
    return temp;
}
```

4. Given a base (denoted by n) and a coefficient (denoted by k), where n,k are integers, write a recursive method called `exponent(int, int)` that returns $n^k$. You can assume that $n > 1$.

Answer:
```java
public int exponent (int n, int k){
    //base case.
    if(k == 0) {
        return 1;
    }
    return n * exponent(n,k-1);
}
```
5. What is the default size if you do not specify a size when you declare an 
ArrayList?
10

6. Given an arbitrary integer value, print recursively all odd numbers down till 1. 
Ex. Given 5, your program should print:
5
3
1
1
You can assume k > 0

Answer:
Public void printOddNumbers (int k){
//take care of the base case
if(k==1){
    System.out.println(k);
    Return;
}

//if it is an even number, subtract 1 to make it odd
if((k%2) == 0)
Return printOddNumbers(k-1);

//we got an odd number.
Else{
    System.out.println(k);
    //subtract 2 to continue making it odd
    Return printOddNumbers(k-2);
}
}
7. Given a string, recursively compute a new string where all the lowercase 'x' chars have been moved to the end of the string.

endX("xxre") → "rexx"
endX("xxhixx") → "hixxxx"
endX("xhixhix") → "hihixxx"

Answer:
public String endX(String str) {
    if (str.length() <= 1) // base case.
        return str;
    if (str.charAt(0) == 'x') // if it is x, recurse but at it at the end
        return endX(str.substring(1, str.length())) + 'x';
    else
        return str.charAt(0) + endX(str.substring(1)); // else, return the first letter, but recurse what's after that.
}

8. Given the following block of code, What does the following code print?

Stack stack = new Stack();
Queue queue = new Queue();

queue.enqueue(a);
queue.enqueue(b);
queue.enqueue(c);
queue.enqueue(d);

stack.push(queue.dequeue());
stack.push(queue.dequeue());
stack.push(queue.dequeue());
stack.push(queue.dequeue());

while (!stack.isEmpty()){
    System.out.println(stack.pop());
}

Answer:
D, C, B, A
9. Short Answer: Which would you use to represent a stack/queue, an array or a linked list? For whichever answer you suggest, give some reasoning as to why you chose an array over a linked list or vice versa.

Answer:
While one can use an array, the argument that one has to dynamically grow the array kinda demonstrates the issue with using an array to build a stack/queue. Arrays give us faster access to any element, but with a stack/queue we don’t want or need to access any particular element, we only need to access the element at the front/top.

The one main advantage that the array has is that is has really fast access to any element within our data, no matter its order/location. With stacks/queues we don’t need to (or really shouldn’t be) accessing any element at any location, we should be accessing only the front/end or the top of the queue/stack. Arrays also have to be dynamically grown.

Linked lists, on the other hand, don’t have to worry about resizing (just create a new node and set its next reference, then adjust top/head). We also shouldn’t ever have to grab an item out of the middle of our list. We don’t have to do any shifting of data after a pop/dequeue either.

10. Given a nonempty linked list, remove the head of the linked list and return the value.

method header is:
public T removeHead(Node head)

Assume we are giving you a getNext method, getData method, setNext method, and setData method

Answer:
public T removeHead(Node head){
    T returnData = head.getData();
    head = head.getNext();
    return returnData;
}
11. Suppose I have the following code for my Node class.

```java
public class Node<E> {
    private E name;
    private Node<E> next;

    public Node (E element) {
        name = element;
        next = null;
    }
}
```

Suppose my linkedList class needs to sort the nodes in alphabetical order. Create a method call insert that will place a new node into the correct order. Make sure that you cover all edge cases and use your generics appropriately. You can assume this is a linked list is not circular and is a singly linked list. You can also assume in linkedList there is a variable that keeps track of the amount of items stored in the list called size. Also assume the compareTo method works correctly as well. Assume there is a head variable. It may or may not be set. Make this method return the index where the node was inserted at. There are to be no duplicates. If the name already exists in the list return -1.

Answer:
```java
public int insert(E newName){
    Node<E> insertMe = new Node(newName);

    //Check if the list is empty
    if (head==null){
        head = insertMe;
    } //end if

    //Bypassing a dummy head
    Node<E> fore = head; Node<E> aft = null;

    //Go until we found the spot to insert or the last real place in the list
    while (fore.getData().compareTo(newName) < 0 && fore != null){
        //If we found a duplicate!
        if (fore.getData().compareTo(newName)==0)
            return -1;
        aft = fore;
        fore = fore.getNext();
    } //end while

    //Now, insert!
    aft.setNext(insertMe);
    insertMe.setNext(fore);
    return 1;
} //end insert
```
12. Now write the compareTo method for the insert problem above. Return 1, if greater than, -1 if less than, or 0 if equal.

Answer: (Please note there are multiple correct solutions. Please check with your SL if you have questions.)

//Since we're sorting things alphabetically, let's assume //our data is strings!
public int compareTo(String other){
    int shortLen;
    //Find the shorter string to avoid //String index out of bounds exceptions in our //loop
    if (this.length() > other.length()){
        shortLen = other.length();
    } else{
        shortLen = this.length();
    }
    String thisStr = this.toLowerCase();
    String otherStr = other.toLowerCase();
    //Go for that short length
    for (int i=0; i < shortLen; i++){}
        //if this is less, return that
        if (thisStr.charAt(i) < otherStr.charAt(i))
            return -1;
        //if other is less, return that
        else if (otherStr.charAt(i) < thisStr.charAt(i))
            return 1;
    //end for
    //Once we made it out of the loop, we either have identical //strings or a string and its substring, so to find out if //they're identical, we need to compare their lengths //So we can just return the length of this - other. If it's //is larger, it will return a positive, if this is smaller //, it will return a negative, and if they're identical, //it will return 0
    return this.length() - other.length();
}
13.

a) What is the difference between a doubly linked list and a singly linked list?

Doubly linked list nodes contain both next and previous fields, so that a node contains a reference to the node before it in the list and to the node after it in the list.

b) Why might you want to use a doubly linked list instead of a singly linked list?

If you ever need to use a list that you can move forwards and backwards in, you would want to use a doubly linked list!

c) Write a method that prepends to the front of a doubly linked list. Don’t forget to take care of any error conditions you can think of! Note: assume the DLLNode class contains the following: getNext(), setNext(), getPrev(), setPrev(), DLLNode(String data).

```java
public class DLLList{
    DLLNode head = new DLLNode(null);

    public void prepend(String data){
        DLLNode addMe = new DLLNode(data);
        addMe.setNext(head.getNext());
        head.getNext().setPrev(addMe);
        head.setNext(addMe);
    }
}
```