CSC 115
Midterm Exam:
Monday, 27 June 2022

Exam duration: 70 minutes

Instructor: Celina Berg

Name: Arfan Hossain (please print clearly!)

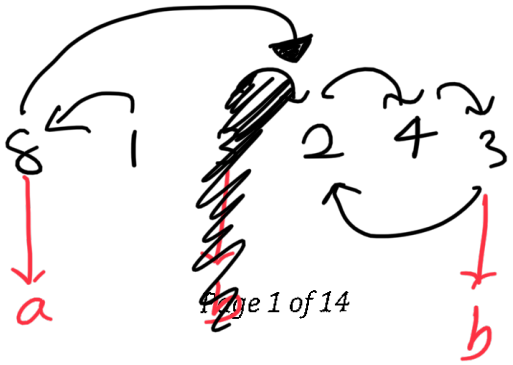
UVic ID number: V00984826 (please print clearly!)

Signature: Arfan Hossain

Students must check the number of pages in this examination paper before beginning to write, and report any discrepancy immediately.

- We will not answer questions during the exam. If you feel there is an error or ambiguity, write your assumption and answer the question based on that assumption.
- Answer all questions on this exam paper.
- The exam is closed book. No books or notes are permitted.
- **Electronic devices, including calculators, are not permitted**.
- The marks assigned to each part are printed within brackets. Partial marks are available.
- There are fourteen (14) pages in this document, including this cover page.
- Pages 5, 8, 9 and 14 are left blank for scratch work. If you write an answer on that page, clearly indicate this for the grader under the corresponding question.
- Clearly indicate only one answer to be graded. Questions with more than one answer will be given a **zero grade**.
- It is strongly recommended that you read the entire exam through from beginning to end before beginning to answer the questions.

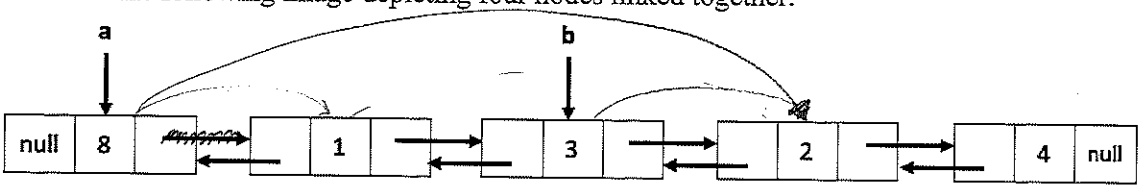
1 8 2 4 3



8 2 4 3 2

Part 1 (10 marks)
For the following questions, write your final answer in the box provided.

a) Consider the following image depicting four nodes linked together.



Next, consider the following code snippet, which updates the state of the nodes shown above:

```
a.next = b.next;
b.prev.next = a;
b.next.next.next = b;
```

After the code shown above is executed, what is the output of the following method which prints out the values of five connected nodes given the call: **`printList(b.prev);`**

```
public static void printList(Node start) {
    Node cur = start;
    for(int i=0; i<5; i++) {
        System.out.println(cur.data + " ");
        cur = cur.next;
    }
    System.out.println();
}
```

Write the integer value you think is printed in the corresponding box.
If you believe **null** is reached before printing 5 values, enter **null** into the box instead of an integer value.

First value printed: [8]

Second value printed: [2]

Third value printed: [null]

Fourth value printed: [null]

Fifth value printed: [null]

b) Recall the `doBracketsMatch` method from lab. Our solution utilized a stack data structure in order to solve the problem by matching close brackets to their corresponding open brackets and checking for an empty stack at the end. What is the **maximum** number of characters that will appear on the stack at any time when our method is called with the `String`: `"(())()(()())"`

Write your final answer in this box: 2

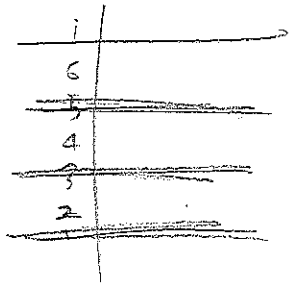
6 5 4 3 2 1

c) Suppose a stack of integers `stk` contains the following 6 elements: 1, 2, 3, 4, 5, 6 (pushed in that order). What is the state of the stack `stk` after the following code snippet has executed? Assume the stack instance methods behave as seen in labs and assignments. 654321

Write your answer in the box provided below as a comma separated list from **left to right** traversing the stack from **top to bottom**.
For example:
the list: 9, 2, 3 would represent a stack with the value 9 at the top, 2 in the middle and 3 at the bottom

```
for(int i=0; i<6; i++) {
    if (i % 2 == 0) {
        stk.push(stk.peek());
        stk.pop();
    }
}
```

6
4
2



Write **only** your final answer in this box:

6, 5, 4, 3, 2, 1

d) Suppose a queue of integers que contains the following 6 elements: 1, 2, 3, 4, 5, 6 (enqueued in that order). What is the state of the queue que after the following code snippet has executed? Assume the queue instance methods behave as seen in labs and assignments.

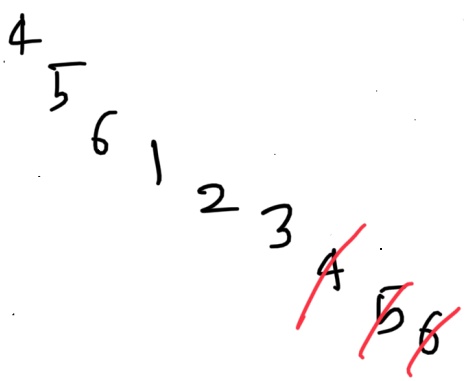
Write your answer in the box provided below as a comma separated list from **left to right** traversing the queue from **front to back**.
For example:
the list: 9, 2, 3 would represent a queue with the value 9 at the front, 2 in the middle and 3 at the back

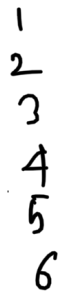
```
for(int i=0; i<6; i++) {
    if (i % 2 == 0) {
        que.enqueue(que.peek());
        que.dequeue();
    }
}
```

Write **only** your final answer in this box:

4, 5, 6, 2, 4, 6

correct: 4,5,6,1,2,3





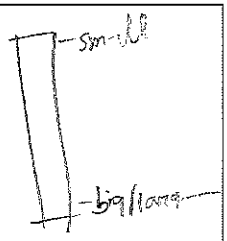
Page left blank intentionally for scratch work if needed...
If you write an answer on this page, clearly indicate this for the grader under the corresponding question.

Part 2 (11 marks)

Consider the interface and an implementation of a stack ADT that holds elements that are `int` type:

```
public interface Stack {
    /*
     * Purpose: returns the boolean state of the stack (empty or not)
     * Parameters: none
     * Returns: boolean - true if stack is empty, false otherwise
     */
    boolean isEmpty();

    /*
     * Purpose: places the given element on the top of the stack.
     * Parameters: int element
     * Returns: void
     */
    void push(int element);

    /*
     * Purpose: removes the value at the top of the stack
     *   and returns it to the caller.
     * Parameters: none
     * Precondition: this Stack is not empty
     * Returns: int - the element that was at the top of the stack
     */
    int pop();

    /*
     * Purpose:  returns the value at the top of the stack,
     *   but does not modify the contents of the stack.
     * Parameters: none
     * Precondition: this Stack is not empty
     * Returns: int - the element at the top of the stack
     */
    int peek();

}// end of Stack interface

public class StackArrayBased implements Stack {
    private static final int INIT_SZ = 10;
    private int[] data;
    private int top;

    /* Constructor
     * Parameters: none
     * Purpose:  initializes data to an int array and top to -1
     */
    public StackArrayBased() {
        data = new int[INIT_SZ];
        top = -1;
    }
    // remainder of this class intentionally omitted
    // you DO NOT need to implement any of the methods for this class

}// end of StackArrayBased class
```

Complete the `isSorted` method below according to the given documentation.




```
/* Method name: isSorted
 * Purpose: This method returns true if the values in stk are
 *      in sorted order, with the smallest value at the top,
 *      otherwise the method will return false.
 *      After the method is complete, stk should be in the
 *      same order it was when the method was called.
 * Parameters: Stack stk - the stack to inspect
 * Precondition: stk is not null
 * Postcondition: stk values and order of values are the same after the call
 * Examples:
 * If stk is empty the method should return true and stk is still empty.
 * If stk contains the values: 2, 4, 4, 8
 *   where 2 is at the top of the stack and 8 is at the bottom,
 *   the method should return true and stk still contains values: 2, 4, 4, 8
 * If stk contains the values: 2, 4, 9, 8
 *   where 2 is at the top of the stack and 8 is at the bottom,
 *   the method should return false and stk still contains values: 2, 4, 9, 8
 * Return type: boolean
 */
//Write the required code below.
// NOTE: you can create additional Stack(s) in your method if needed.
```

```
public boolean isSorted (Stack stk) {
    if ( stk != null) {
        Stack stN1 = new StackArrayBased();
        while (!stk.isEmpty()) {
            int data1 = stk.pop();
            stN1.push (data1);
            if (data1 > stk.peek()){
                return false;
            }
        }
        while(!stN1.isEmpty()) {
            stk.push(stN1.pop());
        }
    }
    return true;
}
```

Page left blank intentionally for scratch work if needed...
If you write an answer on this page, clearly indicate this for the grader under the corresponding question.

Page left blank intentionally for scratch work if needed...
If you write an answer on this page, clearly indicate this for the grader under the corresponding question.

Part 3 (18 marks)

Consider the `Node` and `LinkedList` classes below.
NOTICE: The fields in the `Node` generic class are public.
IMPORTANT: You may **NOT** add fields to either the `Node` or `LinkedList` generic classes.

There are 2 parts to this question:

a) Complete the required `getMiddle` method for the `LinkedList` class according to the documentation given.
Your solution is to be recursive, that is, **any traversals of the list must be done recursively.**
Any solution that contains `for` or `while` loops will result in a **grade of 0.**
You are free to create additional private helper methods in your solution.
Write the required code in the box provided on the following page.

b) Give the efficiency of your `get_middle` method implementation in big-oh notation in the box provided on the following page.

```
public class Node<T> {

    public T       value;
    public Node<T> next;

}// end of Node<T> class


public class LinkedList<T> {

    private Node<T> head;
    private Node<T> tail;

    public LinkedList() {
        head = null;
        tail = null;
    }

    /* Method Name: getMiddle
     *
     * Purpose: returns the middle value in this LinkedList
     *
     * Parameters: none
     *
     * Examples:
     *  If the list is {} the value returned is null
     *  If the list is {1, 2, 3, 4, 5} the value returned is 3
     *  If the list is {6, 7, 8, 9, 6, 3} the value returned is 9
     *
     * Returns:  T - the middle value
     */
     // method implementation to be completed on the following page
```

```
public int getTotal(Node<T> curr, int total) {
    if (curr != null) {
        curr = curr.next;
        return (total + 1);
    } else {
        return total;
    }
}

public Node<T> getNode(Node<T> curr, int pos, int curr) {
    if (curr != null) {
        if (pos == curr) {
            return curr;
        } else {
            curr = getNode(curr.next, pos, ++curr);
        }
    }
}

public T getMiddle() {
    int length = this.getTotal(head, 0);
    if ((length % 2) == 0) {
        int half = length / 2;
        Node<T> curr = getNode(head, half, 0);
        return this.value;
    } else {
        int half = (length / 2) + 1;
        Node<T> curr = getNode(head, half, 0);
        return this.value;
    }
}
```

}// end of LinkedList<T> class

Give the efficiency of your `getMiddle` method in big-oh notation in this box:

Part 4 (15 marks)

Given the following interface:

```
public interface IntegerList {

    /* extend
     *
     * Purpose: adds the values in the given array
     *   to the end of this IntegerList
     *
     * Parameters: int[] array – full array of n values to be added
     *
     * Precondition: array is not null, array is full
     *
     * Returns: nothing
     *
     * Example:
     *  If this list contains the following 3 values in this order:
     *  1, 2, 3
     *  after this method is called with the array {7, 9, 8, 5}
     *  this list will contain the following 7 values in this order:
     *  1, 2, 3, 7, 9, 8, 5
     *
     */
    void extend(int[] array);

} // end of IntegerList interface
```

There are 2 parts to this question:

a) Complete the required method for the `IntegerLinkedList` class beginning below.
Write the required code in the box provided on the following page.
Your solution is to be iterative, that is, **any traversals of the list must be done with `for` or `while` loops.** Any solution that contains recursion will result in a **grade of 0.**

NOTICE: The fields in the `Node` class are public.

IMPORTANT: You may **NOT** add fields to either the `Node` or `IntegerLinkedList` classes.

b) Give the efficiency of your `extend` method implementation in big-oh notation in the box provided on the following page.

```
public class Node {
    public int  value;
    public Node next;
    public Node prev;
}// end of Node class

public class IntegerLinkedList implements IntegerList {

    private Node head;

    public IntegerArrayList() {
        head = null;
    }
```

```
public void extend(int[] array) {
    Node curr = this.head;
    if(head!=null && array.length != 0){
        while(curr!=null){
            curr = curr.next;
        }
        Node tmp = new IntegerArrayList();
        for(int i=0; i<array.length; i++){
            tmp.value = array[i];
            curr.next = tmp;
            tmp.prev = curr;
            curr = currnext;
        }
        currnext = null;
    }
}
```

```
} // end of IntegerLinkedList class
```

Give the efficiency of your **extend** method in big-oh notation in this box:

Page left blank intentionally for scratch work if needed...
If you write an answer on this page, clearly indicate this for the grader under the corresponding question.