INTERNATIONAL SCHOOL OF ENGINEERING

FACULTY OF ENGINEERING

CHULALONGKORN UNIVERSITY

2190221 Fundamental Data Structure and Algorithm

Year 2, Second Semester, Midterm Examination March 02, 2016. Time 08:30-11:30

Name.....Identification no.....No. in CR58.....

Important

1. This exam paper has 4 questions. There are 10 pages in total including this page. The total mark is 82.
2. Write your name, ID, and CR 58 number on top of every page.
3. **All questions asked you to write Java code. If you write in some pseudo code and it is understandable, your score will be deducted appropriately.**
4. **Your answer must only be written in the answering book.**
5. When the exam finishes, students must stop writing and remain in their seats until all question sheets and answering books are collected and the examiners allow students to leave the exam room.
6. A student must sit at his/her desk for at least 45 minutes.
7. A student who wants to leave the exam room early (must follow (5).) must raise his/her hand and wait for the examiner to collect his/her papers. The student must do this in a quiet manner.
8. **No books, lecture notes or written notes of any kinds are allowed in the exam room.**
9. **No calculators are allowed.**
10. A student must not borrow any item from another student in the exam room. If you want to borrow an item, ask the examiner to do it for you.
11. Do not take any part of the question sheet or answering books out of the exam room. All papers are properties of the government of Thailand. Violators of this rule will be prosecuted in a criminal court.
12. A student who violates the rules will be considered as a cheater and will be punished by the following rule:
    - **With implicit evidence or showing intention for cheating, student will receive an F in that subject and will receive an academic suspension for 1 semester.**
    - **With explicit evidence for cheating, student will receive an F in that subject and will receive an academic suspension for 1 year.**

I acknowledge all instructions above. This exam represents **only my own work**. I did not give or receive help on this exam.

Student's signature(.....)

Name……ID……CR58……

You can use methods that you have written in one of the questions in all other questions. Classes and their methods given to you in any of the question are usable in all questions (unless a question forbids their use). Any code not given in this exam must be written out by you.

READ AN ENTIRE QUESTION BEFORE STARTING TO WRITE ANYTHING FOR THAT QUESTION!!!

1. A node for binary search tree (that stores integer) has the following definition:

```
public class BSTNode {
	int data; // value stored in the node.
	BSTNode left; //pointer to lower left BSTNode.
	BSTNode right; //pointer to lower right BSTNode.
	BSTNode parent; //pointer to the BSTNode above.

	public BSTNode(int data){
		this(data,null,null,null);
	}

	public BSTNode(int data, BSTNode left, BSTNode right, BSTNode parent)
	{
		this.data = data;
		this.left = left;
		this.right = right;
		this.parent = parent;
	}

	public static void main(String[] args) {
		BSTNode b = new BSTNode(9);
	}
}
```

A class for binary search tree is:

```
public class BST {
    BSTNode root;
    int size;

    public BST(BSTNode root, int size) {
        this.root = root;
        this.size = size;
    }
}
```

Class BST has the following methods that you can call:

- `TreeIterator findMin(BSTNode n)`: return an iterator marking a node that stores the minimum value in the subtree that has n as its root. If n is null, return null.
- `Iterator find(int v)`: return an iterator marking a node that stores v. If v is not in the tree, return null.
- `boolean isEmpty()`: return true if the tree has no node, false otherwise.
- `Iterator insert(int v)`: add data v into the tree. Existing data in the tree are not added. It returns an iterator marking a node that stores v.

- `void remove(int v): remove v from the tree. If v is not in the tree, do nothing.`

Class **TreeIterator** (it implements interface Iterator) is also available. You can call its following methods:

- ```
  public boolean hasNext();
  ```
- ```
  public boolean hasPrevious();
  ```
- ```
  public int next() throws Exception;
                    // move iterator to the next position,
                  // then returns the value at that position.
  ```
- ```
  public int previous() throws Exception;
                        // return the value at current position,
                        // then move the iterator back one position.
  ```

a. (17 marks) Explain how you can remove a value from a binary search tree. This is a logic behind method **remove**. Draw pictures to help with your explanation.

b. (6 marks) Write code for method **public static BST fixTree(BST x)** of class BST:

- This method receives a tree, **x**, that we are not sure whether it is really a binary search tree.
- It returns a binary search tree that has all its data from **x**, but this time it is really a binary search tree.

2. Assume we are using stack of integer from **class Stack**, which **has the code of all methods** defined in the following Java interface (**class Stack also has a working default constructor**):

```
public interface MyStack {
      public boolean isEmpty();
      public boolean isFull();
      public void makeEmpty();

      //Return data on top of stack.
      //Throw exception if the stack is empty.
      public int top() throws Exception;

      //Remove data on top of stack.
      //Throw exception if the stack is empty.
      public void pop() throws Exception;

      //Add new data on top of stack.
      //Throw exception if the operation is somehow
      //unsuccessful.
      public void push(int data) throws Exception;
}
```

We are using stack in our own **class TestStack**, which is:

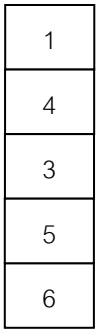
Name……..ID……CR58…….

```
Class TestStack{
        Stack s;
        public Stack oddNumbers(){
                // You have to write code for this method.
        }
}
```

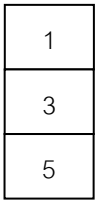
You are to implement method **oddNumbers()**,

- which returns a new stack which has all odd numbers from s, and the numbers are in the same order (from top to bottom) as they are on s.
- when the method finishes its execution, stack s must remain unchanged.

For example, if the original data inside s is



Then the returned stack is



- You do not know the internal workings of s, so you can only use methods provided by MyStack interface.
- You are allowed to create primitive type variables.
- You are not allowed to create non-primitive type variables, or any data structure, except Stack(s).

a. (8 marks) Explain, with illustrated drawing of stack(s), the inner workings of **oddNumbers()**. Your explanation should be clear and step-by-step.

b. (8 marks) Write code for **oddNumbers ()**.

3. Assume we have **Class Queue**, a double-ended queue that stores integers. **This class has a default constructor that you can call**. It also already implements all methods defined in the following interfaces:

```
public interface MyQueue {//each method is assumed to perform in Θ(1)
	//Return the first data.
	//Throw Exception if the queue is empty.
	public int front() throws Exception;

	//Return the last data.
	//Throw Exception if the queue is empty.
	public int back() throws Exception;

	//Remove the first data (return its value too).
	//Throw Exception if the queue is empty.
	public int removeFirst() throws Exception;

	//Insert new data after the last data.
	//Throw exception if the insert fails for some reason.
	public void insertLast(int data) throws Exception;

	//Check if the queue is empty.
	public boolean isEmpty();

	//Check if the queue has no more space to store new data.
	public boolean isFull();

	//Return the number of data currently stored in the queue.
	public int size();
}
```

We are using queues in our own **class TestQueue**, which is:

```
Class TestQueue{
	Queue q1, q2;
	public void crossOver(int p1, int p2){
		// You have to write code for this method.
	}
}
```

You are to implement method **crossOver**, which exchanges values stored from position p1 to q1.size()-1 of q1 with values stored from position p2 to q2.size()-1 of q2. The value of p1 and p2 is assumed to always be from 0 to q1. size()-1 and q2.size()-1 respectively.

For example, if q1 has {1,2,3,4,5} and q2 has {6,7,8,9,10}, when crossOver(2,3) is called, q1 will then store {1,2,9,10} and q2 will store {6,7,8,3,4,5}

- You do not know the internal workings of queue, so you can only use methods provided by the interfaces.

- You are allowed to create primitive type variables.
- You are not allowed to create non-primitive type variables, or any data structure.

a. (12 marks) Draw pictures of queues in various states and explain your implementation of method crossOver.

b. (11 marks) Write code for method crossOver.

c. (4 marks) Show how you analyze asymptotic runtime for your code.

4. Code for implementing a circular doubly-linked list is given below:

```
public interface Iterator {
	public boolean hasNext();
	public boolean hasPrevious();

	public int next() throws Exception;
			// move iterator to the next position,
			// then returns the value at that position.

	public int previous() throws Exception;
				// return the value at current position,
				  // then move the iterator back one position.

	public void set(int value);

}

public class DListIterator implements Iterator {
	DListNode currentNode; // interested position

	DListIterator(DListNode theNode) {
		currentNode = theNode;
	}

	public boolean hasNext() { // always true for circular list.
		return currentNode.nextNode != null;
	}

	public boolean hasPrevious() { // always true for circular list.
		return currentNode.previousNode != null;
	}

	public int next() throws Exception {
		// Throw exception if the next data
		// does not exist.
		if (!hasNext())
			throw new NoSuchElementException();
		currentNode = currentNode.nextNode;
		return currentNode.data;

	}

	public int previous() throws Exception{
		if (!hasPrevious())
			throw new NoSuchElementException();
		int data = currentNode.data;
```

```
                currentNode = currentNode.previousNode;
                return data;
        }

        public void set(int value) {
                currentNode.data = value;
        }
}

class DListNode {
        DListNode(int data) {
                this(data, null, null);
        }

        DListNode(int theElement, DListNode n, DListNode p) {
                data = theElement;
                nextNode = n;
                previousNode = p;
        }

        // Friendly data; accessible by other package routines
        int data;
        DListNode nextNode, previousNode;
}

public class CDLinkedList {
        DListNode header;
        int size;
        static final int HEADERVALUE = -9999999;


        public CDLinkedList() {
                size = 0;
                header = new DListNode(HEADERVALUE);
                makeEmpty();//necessary, otherwise next/previous node will be null
        }

        public boolean isEmpty() {
                return header.nextNode == header;
        }

        public boolean isFull() {
                return false;
        }

        public void makeEmpty() {
                header.nextNode = header;
                header.previousNode = header;
        }
```

```
// put in new data after the position of p.
public void insert(int value, Iterator p) throws Exception {
	if (p == null || !(p instanceof DListIterator))
		throw new Exception();
	DListIterator p2 = (DListIterator) p;
	if (p2.currentNode == null)
		throw new Exception();
```

Need to be filled ! See question 4 a)

```
}

// return position number of value found in the list.
// otherwise, return -1.
public int find(int value) throws Exception {
	Iterator itr = new DListIterator(header);
	int index = -1;
	while (itr.hasNext()) {
		int v = itr.next();
		index++;
		DListIterator itr2 = (DListIterator) itr;
		if (itr2.currentNode == header)
			return -1;
		if (v == value)
			return index; // return the position of value.
	}
	return -1;
}

// return data stored at kth position.
public int findKth(int kthPosition) throws Exception {
	if (kthPosition < 0)
		throw new Exception();// exit the method if the position is
	// less than the first possible
	// position, throwing exception in the process.
	Iterator itr = new DListIterator(header);
	int index = -1;
	while (itr.hasNext()) {
		int v = itr.next();
		index++;
		DListIterator itr2 = (DListIterator) itr;
		if (itr2.currentNode == header)
			throw new Exception();
		if (index == kthPosition)
			return v;
	}
	throw new Exception();
}

// Return iterator at position before the first position that stores value.
// If the value is not found, return null.
public Iterator findPrevious(int value) throws Exception {
	if (isEmpty())
		return null;
	Iterator itr1 = new DListIterator(header);
	Iterator itr2 = new DListIterator(header);
	int currentData = itr2.next();
	while (currentData != value) {
```

```
                        currentData = itr2.next();
                        itr1.next();
                        if (((DListIterator) itr2).currentNode == header)
                                return null;
                }
                if (currentData == value)
                        return itr1;
                return null;

        }

        // remove content at position just after the given iterator. Skip header if
        // found.
        public void remove(Iterator p) {
                if (isEmpty())
                        return;
                if (p == null || !(p instanceof DListIterator))
                        return;
                DListIterator p2 = (DListIterator) p;
                if (p2.currentNode == null)
                        return;
                if (p2.currentNode.nextNode == header)
                        p2.currentNode = header;
                if (p2.currentNode.nextNode == null)
                        return;
                DListIterator p3 = new
DListIterator(p2.currentNode.nextNode.nextNode);
                p2.currentNode.nextNode = p3.currentNode;
                p3.currentNode.previousNode = p2.currentNode;
                size--;
        }

        // remove the first instance of the given data.
        public void remove(int value) throws Exception {
                Iterator p = findPrevious(value);
                if (p == null)
                        return;
                remove(p);
        }

        // remove data at position p.
        // if p points to header or the list is empty, do nothing.
        public void removeAt(Iterator p) throws Exception{
                if (isEmpty() || p == null
                        || !(p instanceof DListIterator)
                        ||((DListIterator) p).currentNode == null
                        || ((DListIterator) p).currentNode == header)
                        return;

                DListIterator p2 = (DListIterator)(findPrevious(p));
                remove(p2);

        }

        // Print each contact out, one by one.
        // To be completed by students.
        public void printList() throws Exception {
                Iterator itr = new DListIterator(header);
                while (itr.hasNext()) {
                        Object data = itr.next();
```

```
                System.out.println(data);

            }
        }

        public int size() throws Exception {
            return size;
        }

        //return iterator pointing to location before position.
        public Iterator findPrevious(Iterator position) throws Exception {
            if (position == null)
                return null;
            if (!(position instanceof DListIterator))
                return null;
            if (((DListIterator) position).currentNode == null)
                return null;

            DListIterator p = ((DListIterator) position);
            DListIterator p2 = new DListIterator(p.currentNode.previousNode);
            return p2;

        }

}
```

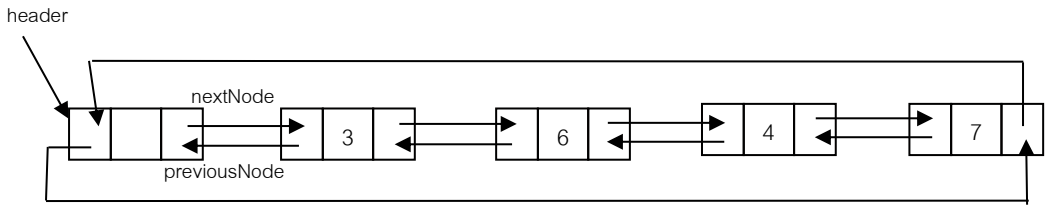
Assume your list always store non-duplicated integers:

a. (6 marks) Fill in the code for method **insert** on page 8. Write your answer in the answer book.

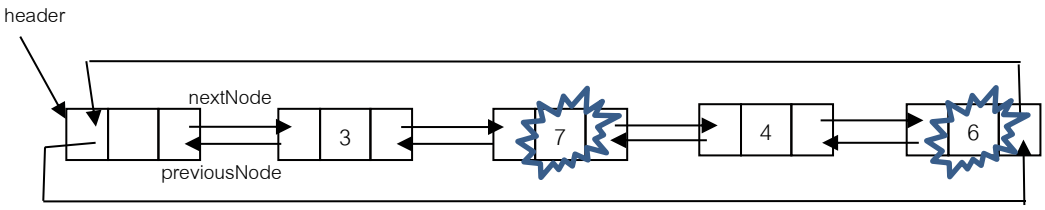
b. (7 marks) Write code for method **public void swap(int v1, int v2)** of class **CDLinkedList**:

- This method swaps v1 and v2.
- If either v1 or v2 is not in the list, simply do nothing.

For example, if the original **this** list is:



then calling swap(7,6) will change **this** to:



c. (3 marks) Show how you analyze the asymptotic runtime of your code in part b.