INTERNATIONAL SCHOOL OF ENGINEERING

FACULTY OF ENGINEERING

CHULALONGKORN UNIVERSITY

2190221 Fundamental Data Structure and Algorithm

Year 2, Second Semester, Midterm Examination March 08, 2018. Time 08:30-11:30

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

<u>Important</u>

1. This exam paper has 4 questions. There are 9 pages in total including this page. The total mark is 40.

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 answer 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 <u>you have written</u> in one of the questions in all other questions. Classes and their methods given to you in a question are <u>NOT</u> usable in other questions. Any code not given in this exam must be written by you.

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

1. (6 marks) A node for binary search tree (that stores integer) has the following definition:

```java
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.
}
```

A class for binary search tree is:

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

      public BST() {
            root = null;
            size = 0;
      }
      …
      …
}
```

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.

Class **TreeIterator** (it implements interface Iterator) is also available.

```java
public class TreeIterator implements Iterator {
      BSTNode currentNode;
            …
            …
}
```

You can call **TreeIterator's** methods as follows:

- **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.

Write code for method **public BST greaterThan(int v)** of class BST:

- This method receives a value, **v**.

- It returns a binary search tree that has all its data from **this**, but with values smaller than or equal to v removed.

- this must remain unchanged after the method is called.

- The use of array, linked list, stack, and queue are forbidden. If you use one of them, you will not get any mark. You are only allowed to create and use binary search tree(s).

2. (10 marks) Assume we are using stacks of integer, s1 and s2, from **class Stack**, which has the code of all methods defined in the following Java interface

```
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;
}
```
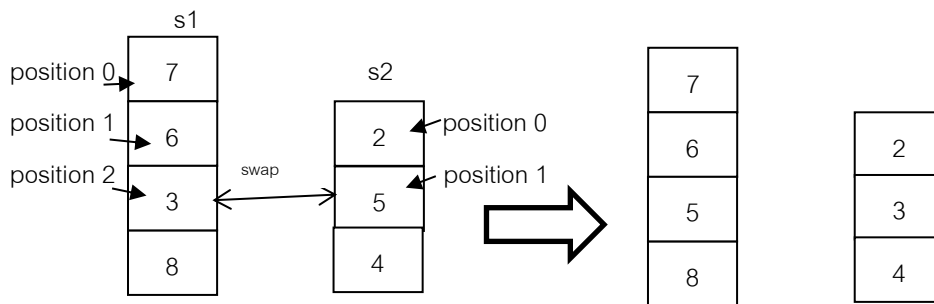
Explain in detail how you can exchange data between position p1 in s1 and position p2 in s2 (Do not write the code. All operations must use methods of Stack).

For example, swapping position 2 of s1 and position 1 of s2 will yield the following:

s1

- You are allowed to create primitive type variables.

- You are not allowed to create non-primitive type variables, arrays, lists, or any data structures, except Stack(s). You get no mark if you do not follow this restriction.

3. (7 marks) In implementation of queue using array, you are given class QueueArray below:

```java
public class QueueArray {
    protected int[] theArray;
    protected int size; // number of currently stored data.
    protected int front;
}
```

Variable **front** is used to mark the position of the first data (data go round the array).

Write code for method **public void insertLast(int x)** of class QueueArray. This method adds x as the last data in the queue.

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

```java
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;
```

```java
        }

        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;
        }
```

```java
public boolean isFull() {
       return false;
}

public void makeEmpty() {
       header.nextNode = header;
       header.previousNode = header;
       size =0;
}

// 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));
```

```java
        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;

}

}
```
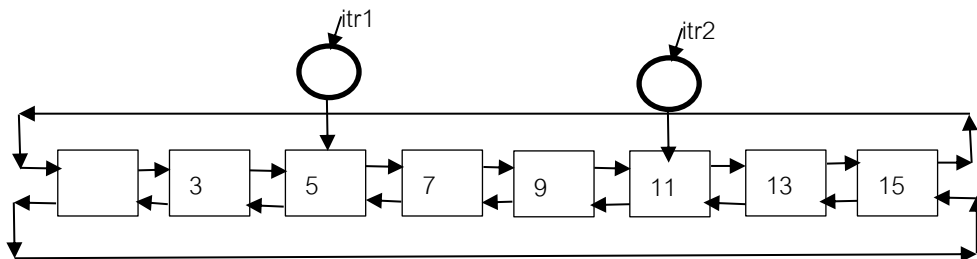
Assume your list always store non-duplicated integers:

    a.   (5 marks) Fill in the code for method **insert** on page 6. Write your answer in the answer book.

    b.   (6 marks) Assume we have a generic non-empty list that has 2 DListIterators, itr1 and itr2, where itr1

        is always to the left of itr2. Both itr1 and itr2 are not at header node. An example list is shown below:

- Explain what you need to do to remove all data from position marked by itr1 upto and including itr2.

  From the example picture, the list after the removal will contain 3, 13, 15.

  - All removed data must be completely cut off from your result list. That is, the removed nodes
    must not be able to access node(s) in the result list (and vice versa).

- Write down the runtime of your explained solution.

c. (6 marks) Write code for method **public void changeToIntersect(CDLinkedList list2)** of class **CDLinkedList**:

   - This method changes **this** list to be a list that contains only data that are both in **this** and in **list2**.
      - <u>You are not allowed to create array, list, stack, queue, or tree, or any data structure. You get 0 mark if you do not follow this restriction.</u>

For example, if the original **this** list contains 2,4,6,8 and list2 contains 4,3,6,9, then your **this** list after the method finishes will contain 4,6.

- Write down the runtime of your explained solution.